Pengembangan Aplikasi Perhitungan Nilai *Understandability* Berdasarkan Rancangan Perangkat Lunak

SKRIPSI

Diajukan untuk memenuhi sebagian persyaratan
mencapai gelar Sarjana Komputer

Disusun oleh:
Mochammad Adhy
NIM: 145150207111033

PROGRAM STUDI TEKNIK INFORMATIKA
JURUSAN TEKNIK INFORMATIKA
FAKULTAS ILMU KOMPUTER
UNIVERSITAS BRAWIJAYA
MALANG
2018

PENGESAHAN

PENGEMBANGAN APLIKASI PERHITUNGAN NILAI *UNDERSTANDABILITY* BERDASARKAN RANCANGAN PERANGKAT LUNAK

SKRIPSI

Untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Komputer

Disusun oleh:
Mochammad Adhy
NIM: 145150207111033

Skripsi ini telah diuji dan dinyatakan lulus pada
19 Juli 2018
Telah diperiksa dan disetujui oleh:

Pembimbing I

Bayu Priyambadha, S.Kom., M.Kom.
NIP. 198209092008121004

Pembimbing II

Fajar Pradana, S.ST., M.Eng.
NIP. 198711212015041004

Mengetahui
Ketua Jurusan Teknik Informatika

Tri Astoto Kurniawan, S.T, M.T, Ph.D
NIP. 197105182003121001

PERNYATAAN ORISINALITAS

Saya menyatakan dengan sebenar-benarnya bahwa sepanjang pengetahuan saya, di dalam naskah skripsi ini tidak terdapat karya ilmiah yang pernah diajukan oleh orang lain untuk memperoleh gelar akademik di suatu perguruan tinggi, dan tidak terdapat karya atau pendapat yang pernah ditulis atau diterbitkan oleh orang lain, kecuali yang secara tertulis disitasi dalam naskah ini dan disebutkan dalam daftar pustaka.

Apabila ternyata didalam naskah skripsi ini dapat dibuktikan terdapat unsurunsur plagiasi, saya bersedia skripsi ini digugurkan dan gelar akademik yang telah saya peroleh (sarjana) dibatalkan, serta diproses sesuai dengan peraturan perundang-undangan yang berlaku (UU No. 20 Tahun 2003, Pasal 25 ayat 2 dan Pasal 70).

Malang, 19 Juli 2018

Mochammad Adhy

NIM: 145150207111033

KATA PENGANTAR

Puji syukur kehadirat Allah SWT yang telah melimpahkan rahmat, taufik dan hidayah-Nya sehingga laporan skripsi yang berjudul "Pengembangan Aplikasi Perhitungan Nilai Understandability Berdasarkan Rancangan Perangkat Lunak" ini dapat terselesaikan.

Penulis menyadari bahwa skripsi ini tidak akan berhasil tanpa bantuan dari beberapa pihak. Oleh karena itu, penulis ingin menyampaikan rasa hormat dan terima kasih kepada:

1. Bapak Bayu Priyambadha, S.Kom., M.Kom. dan Bapak Fajar Pradana, S.ST., M.Eng. selaku Pembimbing skripsi yang telah dengan sabar membimbing dan mengarahkan penulis sehingga dapat menyelesaikan skripsi ini,
2. Bapak Agus Wahyu Widodo, S.T, M.Cs selaku Ketua Program Studi Teknik Informatika,
3. Bapak Tri Astoto Kurniawan, S.T., M.T, Ph.D. selaku Ketua Jurusan Teknik Informatika,
4. Bapak Herman Tolle, Dr.Eng., S.T, M.T. dan Bapak Wayan Firdaus Mahmudy, S.Si, M.T, Ph.D selaku dosen Penasihat Akademik yang selalu memberikan nasehat kepada penulis selama menempuh masa studi,
5. Papa Abdullah Malawat dan Mama Wartini selaku orang tua penulis, Dwi Rama Malawat selaku saudara dan I Gusti Ayu Putri Diani tersayang yang telah memberikan nasihat, kasih sayang, dan perhatian di dalam mendidik dan membantu penulis, serta yang senantiasa tiada henti hentinya memberikan doa dan semangat demi terselesaikannya skripsi ini,
6. Seluruh teman-teman *Ngoding Bitch* Sendi, Danang, Iqbal, Adit, Bambang, Josua, dan Ayi, Kelas C Informatika, BIOS dan khususnya Departemen Artmedia BIOS Devine dan Exalt yang banyak memberi nasihat, dukungan, menemani dalam suka maupun duka, dan menjadi keluarga selama penulis menempuh studi di Informatika Universitas Brawijaya dan selama penyelesaian skripsi ini.

Penulis menyadari bahwa dalam penyusunan skripsi ini masih banyak kekurangan, sehingga saran dan kritik yang membangun sangat penulis harapkan. Akhir kata penulis berharap skripsi ini dapat membawa manfaat bagi semua pihak yang menggunakannya.

Malang, 1 Juli 2018

Penulis
Email: adhymch@gmail.com

ABSTRAK

Mochammad Adhy, Pengembangan Aplikasi Perhitungan Nilai *Understandability* Berdasarkan Rancangan Perangkat Lunak

Pembimbing: Bayu Priyambadha, S.Kom., M.Kom. dan Fajar Pradana, S.ST., M.Eng.

Understandability adalah parameter kualitas perangkat lunak mengenai tingkat kemudahan dalam memahami sebuah modul perangkat lunak yang dikembangkan. Sebagai contoh pentingnya *understandability*, dalam sebuah percobaan mengenai inspeksi kode, 60% dari isu yang dilaporakan oleh *reviewer* profesional pada *maintenance* terkait dengan *understandability*. Oleh karena itu penelitian ini dimaksudkan untuk mengembangkan sebuah aplikasi yang dapat mengukur tingkat *understandability* pada fase perancangan perangkat lunak secara otomatis. Pengukuran *understandability* dilakukan sesuai dengan *multivariate understandability metric*. Pengembangan dilakukan menggunakan metode *waterfall*. Dari hasil penelitian, 100% kebutuhan dapat tervalidasi. Untuk pengukuran efisiensi, diperoleh sistem mampu menyelesaikan 1 pekerjaan pengukuran dalam kurun waktu 10 detik, dan 100% pengguna dapat menyelesaikan tugasnya. Angka tersebut membuktikan sistem berhasil secara efisien membantu mengukur *understandability*. Sedangkan untuk hubungan hasil pengukuran *understandability* dari sistem dengan nilai *maintainability* yang telah diketahui menunjukan korelasi *spearman's rank* sebesar 0.987. Hal tersebut menandakan hasil dari sistem dapat dijadikan salah satu acuan untuk memperkirakan usaha melakukan *maintenance* yang dapat dilakukan sedini mungkin.

Kata kunci: rekayasa perangkat lunak, *maintenance*, *understandability*

ABSTRACT

Mochammad Adhy, ***Development of Understandability Measurement Application Based on Software Design***

***Advisor*: Bayu Priyambadha, S.Kom., M.Kom. dan Fajar Pradana, S.ST., M.Eng.**

Understandability is a software quality parameters that shows the level of ease in understanding a developed software module. As an example of the importance of understandability, in an experiment on code inspection, 60% of issues reported by professional reviewers on maintenance are related to understandability. Therefore this research is intended to develop an application that can measure the level of understandability automatically in the software design phase. Measurement of understandability is done in by using multivariate understandability metric. The development is done by adopting the waterfall model methodology. Based on the results, 100% of requirements are validated. As for the system efficiency, system was able to complete 1 measurement task within 10 seconds, and the user can complete 100% of the task, which proved that system is capable to measure understandability efficiently. As for the relationship between understandability results shows by the system and maintainability that has been known shows the spearman's rank correlation of 0.987, which pointing the results of system can be used as one of the reference to estimating the effort to performing maintenance in early stage of software development.

Keyword: software engineering, maintenance, understandability

DAFTAR ISI

DAFTAR TABEL

DAFTAR GAMBAR

DAFTAR DIAGRAM

BAB 1 PENDAHULUAN

1.1. Latar belakang

Komputasi *modern* dan sistem elektronik biasanya terdapat perangkat lunak yang signifikan di dalamnya. Lebih dari 2 dekade, tingkat kebergantungan masyarakat modern terhadap perangkat lunak selalu meningkat, hal ini menghasilkan permintaan terhadap perangkat lunak yang dapat diandalkan juga meningkat. Karakteristik dari perangkat lunak *object oriented* (OO) seperti misalnya *cohesion* dan *coupling* mempengaruhi tingkat kehandalan perangkat lunak tersebut. *Cohesion* misalnya memberikan dampak positif terhadap tingkat kehandalan seperti kemudahan dalam pemahaman, *reusuable* dan mengurangi *error prone* disisi lain memberi dampak negatif terhadap kompleksitas (Yadav & Khan, 2011). Oleh karena itu, dalam siklus hidup sebuah perangkat lunak kita perlu menjaga kualitas perangkat lunak yang dikembangkan. Penjagaan kualitas dilakukan agar perangkat lunak menjadi lebih bernilai, dapat digunakan dengan baik, tepat sasaran dan mendukung tujuan dari organisasi terkait.

Dalam industri perangkat lunak untuk menjamin dan mengontrol kualitas perangkat lunak membutuhkan alat ukur atau metrik perangkat lunak baik *control metric* maupun *predictive metric*. *Control metric* digunakan untuk mendukung proses manajemen dan *predictive metric* digunakan untuk membantu memprediksi karakteristik dari perangkat lunak (Chidamber & Kemerer, 1994). Salah satu parameter pengukuran kualitas perangkat lunak adalah *understandability* yaitu seberapa tinggi tingkat kemudahan dalam memahami sebuah modul perangkat lunak yang dikembangkan. *Understandability* dapat diukur dengan menghitung derajat kompleksitas, *cohesion* dan *coupling* dari perangkat lunak (Nazir, Khan, & Mustafa, 2010).

Parameter kualitas *understandability* berkaitan dengan *maintainability* pada proses *maintenance*. *Maintenance* yaitu tahap untuk menjaga kelangsungan hidup perangkat lunak dengan melakukan modifikasi pada perangkat lunak tersebut dan komponennya setelah ditemukannya kesalahan yang harus dibenahi, atau karena munculnya alasan untuk beradaptasi sesuai perubahan lingkungan. Dalam melakukan *maintenance,* 80% jatah waktu digunakan untuk memahami sebuah perancangan atau modul perangkat lunak dan dokumen terkait (Izadkhah & Hooshyar, 2017). Hal ini dikarenakan dalam praktiknya tidak selalu tim pengembang yang sama yang melakukan perbaikan kesalahan pada perangkat lunak. Jika pengembang sebelumnya tidak ada maka pengembang yang baru atau staff *maintenance* perlu untuk memahami sistemnya terlebih dahulu. Jika sistem

sulit dipahami, perubahan yang akan dilakukan bisa saja mengakibatkan kesalahan yang serius dan rentetan perubahan. Hal tersebut dapat menghabiskan banyak biaya dan waktu. Sebagai contoh pentingnya *understandability*, dalam sebuah percobaan mengenai inspeksi kode, 60% dari isu yang dilaporakan oleh *reviewer* profesional pada *maintenance* terkait dengan *understandability* (Uchida & Shima, 2004).

Oleh karena itu pada saat pengembangan perangkat lunak, alangkah baiknya jika perancangan model perangkat lunak yang dikembangkan mampu menangani cepatnya perubahan yang terjadi di lingkungan dengan memiliki fleksibilitas dan *understandability* yang baik. Gagal dalam merekayasa perancangan produk perangkat lunak dengan memiliki karekteristik yang fleksibel dan *understandable* ditambah dengan jadwal yang ketat dapat mempengaruhi *on time delivery*, meningkatkan potensi munculnya kesalahan, tingginya harga dalam melakukan *maintenance*, dan pada akhirnya menurunkan tingkat kepuasan pelanggan. Dikhususkan pada tahap *maintenance,* rancangan perangkat lunak dengan *understandability* yang baik dapat memudahkan dan menghemat waktu pembacaan kode sumber program dan dokumen-dokumen terkait (Nazir, Khan, & Mustafa, 2010).

Berdasarkan realita yang telah dijabarkan kualitas *understandability* perlu dijaga. Maka dari itu penulis menawarkan pengembangan kakas bantu SUMIT (*Software Understandability Measurement Instant Tool*) yang dapat mengukur tingkat *understandability* pada fase perancangan perangkat lunak secara otomatis sebagai pengetahuan awal bagi tim pengembang mengenai seberapa besar tingkat *understandability* dari rancangan perangkat lunak yang telah dibuat. Dengan bantuan dari hasil pengukuran menggunakan SUMIT, tim pengembang dapat menghemat waktu dalam melakukan pengukuran *understandability* dan diharapkan dapat mencegah potensi terjadinya kesulitan dan memprediksikan seberapa lama waktu yang dibutuhkan untuk melakukan proses *maintenance* berdasarkan *understandability.* Pengukuran *understandability* pada penulisan kali ini menggunakan karkateristik OO yaitu *inheritance, coupling* dan *cohesion* dari sebuah rancangan perangkat lunak sesuai dengan persamaan *multivariate undestandability metric* (Nazir, Khan, & Mustafa, 2010).

1.2. Identifikasi masalah

Pentingnya paremeter kualitas *understandability* mengakibatkan dibutuhkannya alat bantu ukur yang efisien untuk mengukur tingkat *understandability* pada tahap awal pengembangan perangkat lunak sebagai salah satu bentuk usaha mencegah kesulitan pada tahap *maintenance*.

1.3. Rumusan masalah

Berdasar dari latar belakang yang telah diuraikan, maka penulis merumuskan masalah yang diharapkan dapat diselesaikan melalui penulisan ini sebagai berikut:

1. Bagaimana hasil analisis kebutuhan, perancangan, implementasi, dan pengujian aplikasi SUMIT (*Software Understandability Measurement Instant Tool*) yang menerapkan *multivariate undestandability metric*?
2. Bagaimana efisiensi kinerja aplikasi SUMIT (*Software Understandability Measurement Instant Tool*) terhadap perhitungan manual persamaan *multivariate undestandability metric*?
3. Bagaimana korelasi hasil *understandability* dari SUMIT (*Software Understandability Measurement Instant Tool*) terhadap usaha dalam melakukan *maintenance*?

1.4. Tujuan

Dari latar belakang masalah yang telah disebutkan pada sub-bab 1.1 adapun tujuan dari penulisan ini adalah sebagai berikut:

1. Mengembangkan aplikasi kakas bantu yang dapat membantu pengembang perangkat lunak dalam mengukur secara otomatis nilai *understandability* pada fase awal pengembangan berdasarkan rancangan perangkat lunak.
2. Meningkatkan efisisensi kerja dalam melakukan pengukuran kualitas *understandability* sebuah perangkat lunak.
3. Memberikan kakas bantu untuk melakukan pengukuran tingkat *understandability* sebuah rancangan perangkat lunak agar tim pengembang dapat memprediksikan besar usaha dalam melakukan *maintenance* perangkat lunak berdasarkan tingkat *understandability*.

1.5. Manfaat

Pada penulisan ini, diharapkan dapat memberikan manfaat baik bagi penulis maupun tim pengembang yaitu dapat mengetahui tingkat *understandability* terhadap sebuah perangkat lunak sebagai salah satu parameter pengukuran kualitas perangkat lunak dan dapat menjadi salah satu acuan bagi pihak pengembang perangkat lunak untuk memperkirakan lama waktu dan *cost* yang dibutuhkan dalam tahap *maintenance* dari model perangkat lunak yang dibuat. Serta dengan menilai dan mendeteksi secara dini tingkat *understandability* pada *design* perangkat lunak diharapkan dapat mencegah terjadinya kesulitan pada fase akhir pengembangan yaitu *maintenance* perangkat lunak.

1.6. Batasan Masalah

Dengan berdasarkankepada permasalahan dimana sebelumnya telah dirumuskan, maka hal-hal yang berkaitan dengan sistem akan diberi batasan masalah sebagai berikut:

1. Standar struktur XML yang digunakan sesuai dengan standar *simple format* dari Visual Paradigm.
2. Rancangan perangkat lunak yang diukur menggunakan pendekatan berbasis objek dan merupakan buatan pengembang lain berupa *class diagram* yang bersifat *open source*.
3. Racangan perangkat lunak diukur dalam ruang lingkup sistem bukan komponen.
4. Metrik pengukuran yang digunakan adalah NAssoc *(Number of Association)* untuk mengukur *coupling,* NA (*Number of Attribute*) untuk mengukur tingkat *cohesion,* NOC *(Number of Children)* untuk mengukur *inheritance*, dan *multivariate understandability metric* untuk mengukur understandability .
5. Penelitian dilakukan hanya untuk mengukur tingkat *understandability* tanpa memberikan batasan antara tingkat *understandabilty* yang baik dan buruk.
6. Efisiensi dari otomatisasi pekerjaan pengukuran *understandability* diukur berdasarkan *time based efficiency* (efisiensi waktu) dan *overall relative efficiency* (efisiensi secara keseluruhan) yang dilakukan manusia/pengguna pada saat menggunakan SUMIT.

1.7. Sistematika Pembahasan

Untuk memenuhi tujuan yang diharapkan oleh penulis, maka sistematika penulisan yang akan disusun dalam tugas akhir ini adalah sebagai berikut :

BAB 1 Pendahuluan

Berisi mengenai latar belakang penulisan, identifikasi masalah, rumusan masalah, batasan masalah, tujuan penulisan, manfaat penulisan, dan sistematika penulisan.

BAB 2 Landasan Kepustakaan

Berisi teori-teori yang akan menguraikan tentang proses menjamin kualitas dan mengontrol kualitas pada perangkat lunak, tahap *maintaining* pada perangkat lunak, metodologi pengembangan, library yang dipakai selama pengembangan dan metrik-metrik yang berkaitan dengan penulisan ini seperti *understandability*

metric, dan *object-oriented metric* diantaranya yang berkaitan dengan *cohesion*, *coupling*, dan *inheritance*.

BAB 3 Metode Penelitian

Berisi mengenai metode penelitian yang akan dipakai. Dalam penelitian ini langkah-langkah yang diambil yaitu studi kepustakaan, rekayasa kebutuhan sistem, perancangan sistem, implementasi, pengujian dan analisis hasil yang meliputi pengujian unit, pengujian integrasi, pengujian validasi dan pengujian efisiensi, analisis hasil dilakaukan untuk melihat korelasi antara hasil *understandability* yang diperoleh sistem terhadap *maintainability*, dan membuat kesimpulan dan saran dari penulisan. Sedangkan untuk pengembangan sistem yang dibuat mengadopsi *SDLC Waterfall Model*.

BAB 4 Rekayasa Kebutuhan

Berisi mengenai deskripsi umum sistem, identifikasi aktor dan spesifikasi kebutuhan sistem yang dibagi kedalam 2 jenis kebutuhan yaitu kebutuhan fungsional dan non-fungsional, kebutuhan eksternal sistem, batasan sistem, lingkungan pengembangan sistem dan lingkungan operasi sistem dengan pedekatan berorientasi objek. Pemodelan kebutuhan akan menggunakan *use case diagram* dan diuraikan lebih terperinci menggunakan *use case Scenario*.

BAB 5 Perancangan Sistem dan Implementasi

Berisi mengenai bagaimana sistem dirancang dan implementasinya. Perancangan sistem terdiri dari perancangan arsitektur yang dimodelkan menggunakan *sequence diagram* dan *class diagram*, perancangan algoritma, deskripsi API/*library* yang digunakan serta perancangan antarmuka pengguna. Pada bagian implementasi terdiri dari spesifikasi sistem yang dikembangakan, lingkungan pengembangan, kode sumber, dan hasil implementasi.

BAB 6 Pengujian dan Analisis Hasil

Berisi bagaimana pengujian validasi dilakukan yang terdiri dari 3 buah tahap yaitu pengujian unit, pengujian integrasi, dan pengujian validasi. Pengujian unit dilakukan untuk menguji setiap fungsi yang ada pada sistem apakah telah berjalan dengan baik sesuai algoritma yang dirancang. Pengujian integrasi dilakukan untuk menguji ketika fungsi-fungsi dalam sistem berkolaborasi dalam melakukan sebuah tugas apakah sudah sesuai dengan algoritme yang dirancang. Dan pengujian validasi untuk menguji apakah setiap kebutuhan sistem dari proses rekayasa kebutuhan baik fungsional atau non-fungsional telah diimplementasi dan tersedia

dengan benar. Pada bagian analisis hasil akan diuji seberapa besar korelasi dari hasil *understandability* yang diperoleh sistem terhadap nilai *maintainability* yang telah diketahui dari penelitian sebelumnya.

BAB 7 Penutup

Berisi tentang kesimpulan yang berasal dari hasil penulisan dan saran mengenai pengembangan penulisan dimasa yang akan datang.

BAB 2 LANDASAN KEPUSTAKAAN

Landasan Kepustakaan dalam penulisan ini akan menguraikan tentang proses menjamin kualitas dan kontrol kualitas pada perangkat lunak, fase *maintenance* perangkat lunak, karakteristik *object-oriented design*, *understandability metric*, metodolgi pengembangan, dan *Framework*/API/*library* yang digunakan selama pengembangan.

2.1. Menjamin Kualitas dan Mengontrol Kualitas Perangkat Lunak

Istilah "menjamin kualitas" dan "kontrol kualitas" secara luas digunakan dalam industri manufaktur termasuk industri teknologi informasi. Menjamin kualitas (QA) adalah definisi standar yang seharusnya memimpin kepada produk berkualitas tinggi dalam proses manufaktur. Mengontrol kualitas (QC) adalah aplikasi dari proses menjamin kualitas untuk menyingkirkan produk yang tingkat kualitas tidak diperlukan. Dalam dunia rekaya perangkat lunak, jaminan kualitas berarti definisi prosedur, proses, dan standar yang ditujukan untuk memastikan bahwa kualitas perangkat lunak tercapai. Dalam kasus lain, jaminan kualitas juga mencakup semua manajemen konfigurasi, verifikasi dan validasi kegiatan yang berlaku setelah produk telah diserahkan oleh tim pengembangan. Untuk menjamin dan mengontrol kualitas perangkan lunak diadakanlah manajemen kualitas dimana menyediakan pemeriksaan yang bersifat independen dari proses pengembangan perangkat lunak. Proses manajemen kualitas adalah pemeriksaan proyek yang ditangani untuk memastikan bahwa produk perangkat lunak yang dikembangkan konsisten sesuai dengan standar dan tujuan organisasi terkait (Sommerville, 2011)

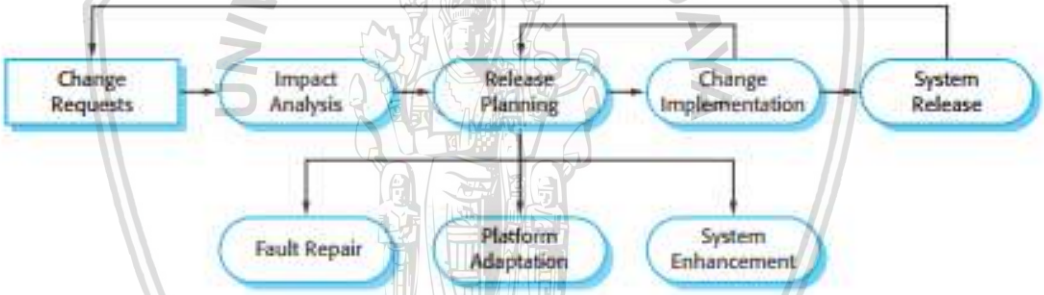
Proses QA sendiri merupakan alat dan metode yang digunakan untuk mendukung penggunaan standar. Standar perangkat lunak itu sendiri ada 3 yaitu (Sommerville, 2011):

1. Standar kebijakan organisasi. Yaitu standar yang didasarkan pada pengetahuan tentang praktik terbaik atau yang paling sesuai untuk perusahaan. Pengetahuan ini paling sering diperoleh setelah banyak *trial and error*. Standar ini membantu perusahaan untuk menghindari kesalahan yang tejadi sebelumnya.
2. Standar kerangka kerja yang mencerminkan ekspektasi pengguna. Karena kualitas perangkat lunak bersifat subjektif dan dengan menggunakan standar kita dapat mendirikan dasar untuk memutuskan jika tingkat kualitas yang diperlukan telah dicapai.
3. Standar dalam kontinuitas. Ketika pekerjaan yang dilakukan oleh satu orang mengambil dan dilanjutkan oleh orang lain Standar ini

memastikan bahwa setiap *engineer* dalam sebuah organisasi mengadopsi praktik-praktik yang sama.

Dalam menjamin dan mengontrol kualitas perangkat lunak akan melibatkan metrik perangkat lunak baik *control metric* atau *predictive metric*. Seperti namanya, *control metric* mendukung proses manajemen, dan *predictive metric* membantu memprediksi karakteristik dari perangkat lunak. *Control metric* atau metrik kendali biasanya berhubungan dengan proses perangkat lunak. Contoh dari *control metric* adalah rata-rata upaya dan waktu yang dibutuhkan untuk memperbaiki kesalahan yang dilaporkan. Sedangkan *predictive metric* terkait dengan perangkat lunak itu sendiri dan kadang-kadang dikenal sebagai "produk metrik". Contoh *predictive metric* adalah *cyclomatic complexity* dari sebuah modul. Pengukuran yang dilakukan pada perangkat lunak dapat digunakan untuk mengumpulkan data kuantitatif mengenai perangkat lunak dan proses perangkat lunak. Dengan menggunakan nilai metrik perangkat lunak yang dikumpulkan kita dapat membuat kesimpulan tentang kualitas produk dan proses rekayasa perangkat lunak (Chidamber & Kemerer, 1994).

2.2. *Maintenance* Perangkat Lunak



Gambar 2.1 Proses Evolusi Perangkat Lunak

Sumber: (Sommerville, 2011)

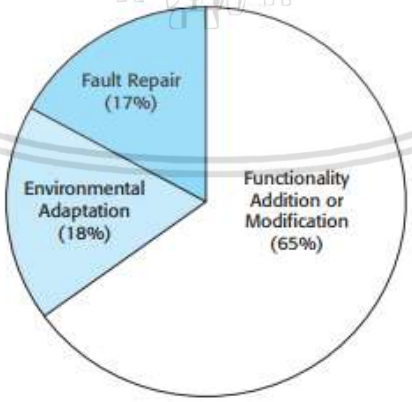
Berdasarkan istilah dari kamus Oxford *maintenance* didefinisikan sebagai berikut: "menjaga sebuah karya dalam kondisi yang tepat". Hal ini mengisyaratkan bahwa *maintenance* adalah tindakan yang harus diambil untuk mencegah kegagalan komponen atau untuk memperbaiki kerusakan pada perangkat agar tetap dapat bekerja dengan baik (Sullivan, 2010). Maintenance merupakan bagian dari evolusi perangkat lunak. Evolusi perangkat lunak dapat dipicu oleh berubahnya kebutuhan bisnis, terdapat laporan mengenai kesalahan pada perangkat lunak, atau perubahan yang diakibatkan oleh sistem lain pada lingkungan perangkat lunak berada. Pada Gambar 2.1 menunjukan Gambaran umum mengenai proses evolusi perangkat lunak. Proses tersebut mengandung aktifitas pokok yaitu analisis perubahan, perencanaan rilis, implementasi perubahan, dan merilis sistem kepada pelanggan (Sommerville, 2011).

Maintenance perangkat lunak adalah proses umum dalam mengubah sistem setelah sistem tersebut diluncurkan. Perubahan yang dibuat pada

perangkat lunak dapat berupa perubahan sederhana seperti memperbaiki coding error, perubahan yang lebih luas untuk memperbaiki kesalahan pada perancangan, atau membuat peningkatan yang signifikan seperti memperbaiki kesalahan pada spesifikasi atau menyediakan kebutuhan baru. Perubahan diimplementasi dengan memodifikasi komponen sistem yang sudah ada dan yang seperlunya dan/atau dengan menambahkan komponen baru pada sistem. *Maintenance* perangkat lunak sendiri dibagi atas tiga jenis yaitu (Sommerville, 2011):

1. *Fault repair*, merupakan *maintenance* yang dilakukan untuk memperbaiki kesalahan yang terjadi. *Coding errors* biasanya lebih murah untuk dikoreksi. Kesalahan pada perancangan lebih mahal untuk dikoreksi karena dapat mengakibatkan penulisan ulang sejumlah komponen program. Kesalahan pada kebutahan merupakan yang paling mahal untuk diperbaiki karena dapat mengakibatkan perancangan ulang sistem yang besar jika diperlukan.
2. *Platform adaptation*, merupakan *maintenance* yang dilakukan ketika beberapa aspek pada lingkungan perangkat lunak mengalami perubahan seperti, perangkat keras, sistem operasi, atau perangkat lunak pendukung lain. Sistem perangkat lunak harus dimodifikasi agar dapat beradaptasi dan mengatasi perubahan lingkungan yang terjadi.
3. *System enhancement/Functionality addition*, merupakan *maintenance* yang diperlukan ketika terjadi perubahan pada kebutuhan sebagai jawaban dari perubahan organisasi atau bisnis. Skala perubahan yang dilakukan pada jenis maintenance ini jauh lebih besar dibanding jenis *maintenance* lainnya.

2.2.1. Relasi Pengembangan dan Maintenance Perangkat Lunak

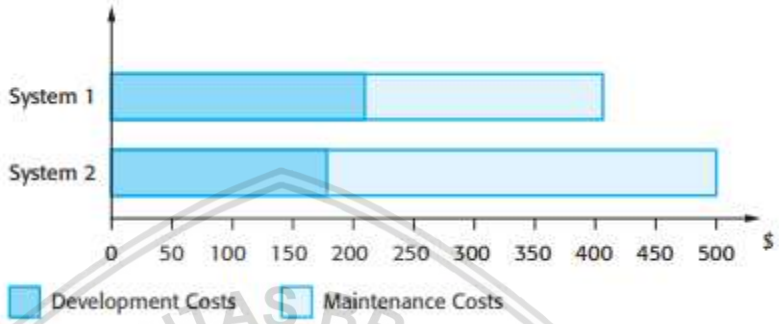


Gambar 2.2 Distribusi Usaha dalam *Maintenance*

Sumber: (Sommerville, 2011)

Terdapat beberapa penulisan pada *maintenance* perangkat lunak yang memperlajari hubungan di antara *maintenance* dengan pengembangan dan dengan aktifitas maintenance yang lain (Krogstie dkk., 2005, Lientz, 1980, Nosek, 1990, Sousa, 1998, Guimaraes, 1983). Karena perbedaan terminologi, secara detail

penulisan-penulisan tersebut tidak dapat dibandingkan namun, secara meluas setuju bahwa biaya relatif maintenance dibandingkan dengan pengembangan sistem baru jauh lebih besar, dan maintenance berupa penambahan kebutuhan baru menghabiskan biaya lebih besar dibandingkan maintenance yang hanya memperbaiki kesalahan. Pada Gambar 2.2 menunjukan perkiraan distribusi biaya maintenance berdasarkan jenisnya. Meskipun perkiraan tersebut sudah lebih dari 25 tahun, distribusi biaya *maintenance* berdasarkan jenisnya tidak sepertinya mengalami perubahan yang signifikan (Sommerville, 2011).



Gambar 2.3 Biaya Pengembangan dan *Maintenance*

Sumber: (Sommerville, 2011)

Penganggaran biaya selalu efektif jika kita menginvestasikannya pada usaha merancang dan mengimplementasi sebuah sistem untuk mengurangi biaya pada perubahan mendatang. Oleh karana itu, pekerjaan pada saat pengembangan untuk membuat perangkat lunak yang mudah untuk dipahami dan diubah dapat mengurangi biaya evolusi. Teknik rekayasa perangkat lunak yang baik seperti ketepatan spesifikasi, penggunaan pengembangan berbasis objek, dan manajemen konfigurasi berkontribusi dalam menurunkan biaya *maintenance*. Gambar 2.3 menunjukan biaya kehidupan rekayasa perangkat lunak secara menyeluruh menurun ketika lebih banyak usaha dihabiskan selama pengembangan sistem untuk menghasilkan sistem yang *maintainable*. Penyebab biaya penambahan fungsionalitas baru pada saat sistem telah beroperasi menjadi lebih mahal yaitu dikarenakan (Sommerville, 2011):

1. Stabilitas tim, hal yang normal bagi tim pengembang untuk dirombak dan bekerja pada proyek baru. Tim baru atau individual baru yang ditugaskan melakukan maintenance tidak memahami sistem atau latar belakang dari keputusan perancangan sistem. Mereka butuh waktu untuk memahami sistem yang sudah ada sebelum mengimplementasi perubahan.
2. Praktik pengembangan yang buruk. Kontrak untuk melakukan maintenance biasanya terpisah dengan kontrak untuk melakukan pengembangan. Maintenance dapat diberikan kepada perusahaan yang berbeda dari pengembang asli sistem. Dengan kata lain pengembang asli bisa saja menggunakan jalan pintas agar sistem dapat selesai tanpa memikirkan perubahan dimasa mendatang.

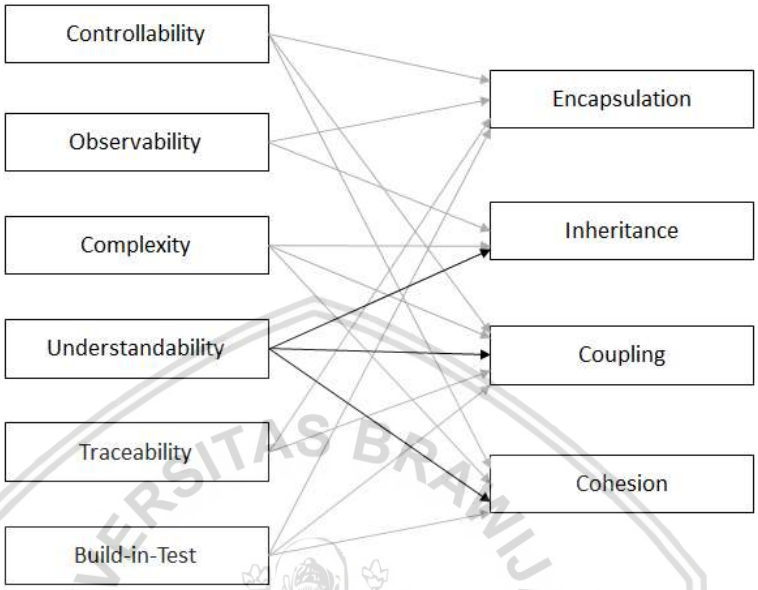
3. Kemampuan staff, dimana pekerjaan maintenance kebanyakan diberikan kepada staff yang lebih muda (junior) yang dinilai masih kurang pengalaman dan tidak familiar dengan domain permasalahan.
4. Umur dan struktur program, seiring berjalannya waktu ketika sebuah program mengalami perubahan, strukturnya akan cenderung merosot. Alhasil dengan bertambahnya umur sebuah program maka program tersebut akan semakin sulit dipahami dan diubah.

Oleh sebab itu untuk meningkatkan *maintainability* dan mengurangi biaya evolusi perangkat lunak di masa mendatang akan lebih baik jika dalam fase perancangan tim pengembang salah satunya harus mampu merancang sistem yang memiliki tingkat *understandability* yang tinggi. Dan untuk membantu hal itu penulis menawarkan aplikasi yang dapat mempermudah perhitungan *understandability* yang dapat dilakukan di fase perancangan perangkat lunak. *Understandability* dipilih karena merupakan kunci awal dalam melakukan *maintenance.*

2.3. Karakteristik Pendekatan Berbasis Objek

Saat ini konsep analisis berbasis objek dan perancangan berbasis objek populer di dalam lingkungan pengembangan perangkat lunak atau sering disingkat dengan sebutan OO (*object oriented*). Metodologi pendekatan berbasis objek (OO) berfokus pada objek sebagai agen primer yang terlibat dalam komputasi. Setiap data dari kelas dan operasi terkait dikumpulkan ke dalam satu entitas (sistem) tunggal. Metodologi ini membutuhkan usaha yang lebih di awal tahap daur hidup pengembangan perangkat lunak untuk mengidentifikasi objek-objek, kelas, atrubut dan operasi serta hubungannya di antara satu sama lain. Kaidah pendekatan berbasis objek memberikan petunjuk kepada *designer* mengenai apa yang harus dilakukan dan apa yang harus dihindari. Sudah ada beberapa pengukuran yang didefinisikan untuk memperkirakan perancangan berbasis objek. Ada beberapa karakteristik esensial OO yang dijadikan tolak ukur pengukuran kualitas perancangan di antaranya, *cohesion*, *coupling*, enkapsulasi, dan *inheritance* (pewarisan). *Cohesion* merupakan hubungan elemen di dalam sebuah komponen yang mengisyaratkan konsistensi internal dari bagian-bagian yang ada pada perancangan. Sebuah kelas memiliki *cohesion* yang baik ketika elemen-elemen di dalam kelas tersebut berkorelasi sangat kuat. *Coupling* merupakan hubungan di antara satu modul ke modul yang lain. *Coupling* mengisyaratkan hubungan atau kebergantungan di antara modul-modul di dalam satu struktur perangkat lunak. *Inheritance* atau pewarisan adalah membagi atribut dan operasi di antara kelas yang berdasar pada hubungan hirarki. Hal ini merupakan mekanisme dimana sebuah objek membutuhkan karakteristik dari satu atau banyak objek lainya. Sedangkan enkapsulasi adalah sebuah mekanisme perwujudan abstraksi data dan menyembunyikan informasi. Enkapsulasi menyembunyikan spesifikasi internal objek dan hanya menampilkan *interface* ekstrernal saja (Nazir, Khan, & Mustafa, 2010).

Dalam penulisan kali ini untuk mengukur *understandability* pada tahap perancangan melibatkan karakteristik OO berupa *cohesion*, *coupling*, dan *inheritance*. Berikut pada Gambar 2.4 menunjukan bagaimana ketiga karekteristik OO berkorelasi dengan *understandability*:



Gambar 2.4 Hubungan *Understandability* dengan Karakteristik OO

Sumber: (Nazir, Khan, & Mustafa, 2010)

Pada Gambar 2.4 menunjukan bahwa *cohesion*, *coupling*, dan *inheritance* merupakan karakteristik OO yang berkaitan dengan *understandability*. Hal ini didapat dari observasi dampak aspek tertentu pada pengujian yang mempengaruhi kemudahan dalam pengerjaanya. Hasilnya disimpulkan bahwa understandability dipengaruhi oleh faktor seperti *cohesion* yang memberikan dampak positif ketika nilainya bertambah sedangkan *coupling* dan *inheritance* memberikan dampak negatif jika nilainya bertambah (Nazir, Khan, & Mustafa, 2010).

2.4. *Understandability* Sebuah Perangkat Lunak

Understandability dari perangkat lunak merupakan salah satu karakter penting dari kualitas perangkat lunak. Hal ini disebabkan *understandability* memiliki pengaruh pada evolusi perangkat lunak seperti *reuse* atau *maintenance*. Kesulitan dalam memahami sistem membatasi *reuseability* ketika pengembang mencoba menggunakan kembali sebuah sistem perangkat lunak dari pengembang lain. Tidak jarang perubahan yang akan dilakukan pengembang lain pada sebuah perangkat lunak membutuhkan penggunaan ulang perancangan atau kode fungsi yang sudah ada untuk melakukan *fault repair*, *platform adaptation*, atau *system enhancement*. Jika pengembang sebelumnya tidak ada maka pengembang yang baru atau staff *maintenance* perlu untuk memahami sistemnya terlebih dahulu.

Jika sistem sulit dipahami, perubahan yang akan dilakukan bisa saja mengakibatkan kesalahan yang serius dan rentetan perubahan. Hal tersebut dapat menghabiskan banyak biaya dan waktu. Sebagai contoh pentingnya *understandability*, dalam sebuah percobaan mengenai inspeksi kode, 60% dari isu yang dilaporakan oleh *reviewer* profesional pada *maintenance* terkait dengan *understandability* (Uchida & Shima, 2004).

Sistem perangkat lunak cenderung untuk menjadi semakin kompleks. Dengan bertambahnya kompleksitas dapat mempengaruhi beberapa atribut kualitas seperti *understandability* dan *maintainability*. Hal signifikan dari *understandability* sudah sangat jelas dan dapat diartikan sebagai berikut "jika kita tidak dapat mempelajari sesuatu, kita tidak dapat memahaminya. Jika kita tidak paham tentang sesuatu, kita tidak dapat menggunakannya. Kita tidak dapat memelihara sebuah sistem yang tidak dapat kita pahami atau tidak mudah melakukannya. Dan kita tidak dapat membuat perubahan pada sistem kita jika kita tidak dapat mengerti bagaimana sistem tersebut akan bekerja ketika perubahan diimplementasikan." Ada beberapa aspek dari artefak pada perangkat lunak yang mempengaruhi *understandability*. Kelas yang besar dan kompleks sulit untuk dimengerti oleh manusia, terutama jika kelas tersebut memiliki *cohesion* yang rendah. Perancangan perangkat lunak yang baik dengan kompleksitas yang terjaga mampu menurunkan *coupling* dan meningkatkan *cohesion* dimana mempermudah *understandability* (Nazir, Khan, & Mustafa, 2010).

2.5. Object Oriented Design (OOD) *Metric*

Dalam dunia pengembangan perangkat lunak ada banyak metrik kualitas yang digunakan dalam pengukuran seperti mengukur *coupling, cohesion, complexity* dll. Namun *metric* seperti dari Chidamber dan Kemerer (C & K) atau Lorenz dan Kidd yang dianggap sebagai tulang punggung dari metrik perancangan berorientasi objek (OOD), yang telah tegas digunakan pada perancangan kelas. Metrik-metrik OOD inilah yang akan digunakan dalam pengukuran kualitas perancangan perangkat lunak. Berikut adalah beberapa metrik pengukuran yang dapat digunakan pada OOD (Genero & Mario, 2001):

Tabel 2.1 Metrik OOD

Metrik	Definisi
NC	Total jumlah *class* dalam sebuah sistem
NA	Total jumlah atribut dalam sebuah sistem
NM	Total Jumlah *method* dalam sebuah sistem
NAssoc	Total Jumlah Asosiasi dalam sebuah sistem
NAgg	Total Jumlah Aggregasi dalam sebuah sistem
NDep	Total jumlah hubungan dependensi

NGen	Total jumlah hubungan generalisasi dalam sebuah sistem
NOC	Total jumlah kelas *children*
NaggH	Total jumlah hirarki aggregasi dalam sebuah sistem
NgenH	Total jumlah hirarki generalisasi dalam sebuah sistem
Max Hagg	Jumlah maksimum dari kedalaman yang dicapai oleh hirarki hubungan aggregasi pada sebuah sistem
Max DIT	Jumlah maksimum tingkat kedalamann *inheritance* dalam sebuah sistem

Berdasarkan Tabel 2.1 untuk mengisi model *multivariate understandability metric* digunakan metrik NA untuk mengitung *cohesion*, metrik NAssoc untuk mengitung *coupling*, dan metrik NOC untuk menghitung *inheritance*.

2.6. Understandability Metric

Dalam perhitungan *understandability* terdapat beberapa metrik yang dapat digunakan namun dipenulisan kali ini penulis hanya menyebut sebagian kecil saja diantaranya, perhitungan *understandability* menggunakan *probabilistic model* yang mengukur understandability berdasarkan jumlah pegawai dan jumlah percobaan yang dilakukan untuk merekonstruksi sistem perangkan lunak hingga benar. Metrik ini bertindak sebagi *control metric* dan menilai bahwa *understandability* tidak hanya dilihat dari sisi produk tetapi dari kemampuan setiap individu yang terlibat (Uchida & Shima, 2004). Metrik selanjutnya yaitu pengukuran *understandability* berdasarkan kompleksitas sebuah kelas atau dikenal dengan *class complexity metric* (CCM). Metrik ini diusulkan untuk mengukur *understandability* pada level kelas. Dalam pengukurannya akan diambil nilai jumlah *cyclometic complexity* dari sebuah kelas, jumlah *method* sebuah kelas, jumlah variabel yang dideklarasi, jumlah *method* eksternal yang dipanggil, jumlah *method* internal yang dipanggil, dan jumlah total baris kode (Rajnish, 2014). Berikutnya metrik *understandability* berdasarkan hirarki *inheritance*. Metrik ini mengjitung *understandability* berdasarkan pewarisan dari *super class* ke *sub class*. Dengan metrik ini kita dapat mengitung *understandability* pada fase perancangan dan mampu menghitung *understandability* dari sebuah kelas atau rataanya dalam satu sistem (Kumar & Prasad, 2015). Metrik lainya yaitu menghitung understandability menggunkan model regresi linear multi-variabel atau lebih dikenal dengan *multivariate understandability metric*. Metrik ini menyusun model persamaan berbentuk regresi linear multi variabel yang menunjukan bentuk korelasi dari variabel terikat terhadap variabel saling bebas. Metrik ini melibatkan *inheritance*, *coupling*, dan *cohesion* sebagai variabel saling bebas dan *understandability* sebagai variabel terikat. Metrik ini digunakan pada level *class diagram* pada fase perancangan. Korelasi dari variabel saling bebas dan varibel terikat pada *multivariate understandability metric* bernilai 0.94 sehingga dinilai memiliki nilai validasi yang tinggi (Nazir, 2010). *Multivariate understandability metric* ini telah

dikembangkan agar dapat mengukur *understandability* pada tingkat *package diagram* dan menghasilkan haril korelasi lebih tinggi antara variabel saling bebas dan variabel terikat yaitu 0.96 (Singh & Tripathi, 2015).

Dalam penulisan kali ini penulis akan menggunakan *multivariate understandability metric* untuk mengukur tingkat understandability pada fase perancangan perangkat lunak yang dinilai sebagai bentuk upaya memudahkan usaha *maintenance* nantinya. Pemilihan *multivariate understandability metric* pada level *class diagram* dikarenakan pada penulisan kali ini pengukuran akan dilakukan dari segi produk dan bukan *control*. Produk disini adalah *class diagram* yang dihasilkan pada fase perancangan serta model metrik ini telah divalidasi memiliki nilai korelasi sangat kuat antara karakteristik OO pada *class diagram* dengan *understandability* sebesar 0.94 berdasarkan *spearman's rank corellation*.

2.6.1. *Multivariate Undestandability Metric*

Multivariate undestandability metric adalah sebuah model regresi linear multi variabel (*multivariate linear model*) yang digunakan untuk mengukur *understandability* berdasarkan beberapa variabel saling bebas diantaranya *coupling, cohesion,* dan *inheritance* yang dinilai mempunyai pengaruh dalam menentukan tingkat *understandability* (variable terikat). Konsep ini digunakan untuk mendapatkan koefisien yang membangun hubungan antara variabel terikat dan variabel saling bebas dimana variabel yang memiliki dampak terhadap *understandability* telah diporsi secara proposional. Dalam model ini nantinya akan dikombinasikan 3 metrik perhitungan yang sesuai dengan atribut atau variabel bebas yang mempengaruhi *understandability*. Ketiga metrik itu adalah *NAssoc* untuk menghitung *coupling*, *NA* untuk menghitung *cohesion*, dan *NOC* untuk menghitung *inheritance* (Nazir, Khan, & Mustafa, 2010).

Berikut ini model umum *multivarite undestandability metric* pada persamaan 1:

$$Understandability = \alpha_0 + \beta_1 \times Coupling + \beta_2 \times Cohesion + \beta_3 \times Inheritance \quad (1)$$

Dimana,

Coupling : Jumlah *coupling* berdasarkan metrik *NAssoc* (jumlah asosiasi)

Cohesion : Jumlah *cohesion* berdasarkan metrik *NA* (jumlah atribut)

Inheritance : Jumlah *coupling* berdasarkan metrik *NOC* (jumlah kelas *children*)

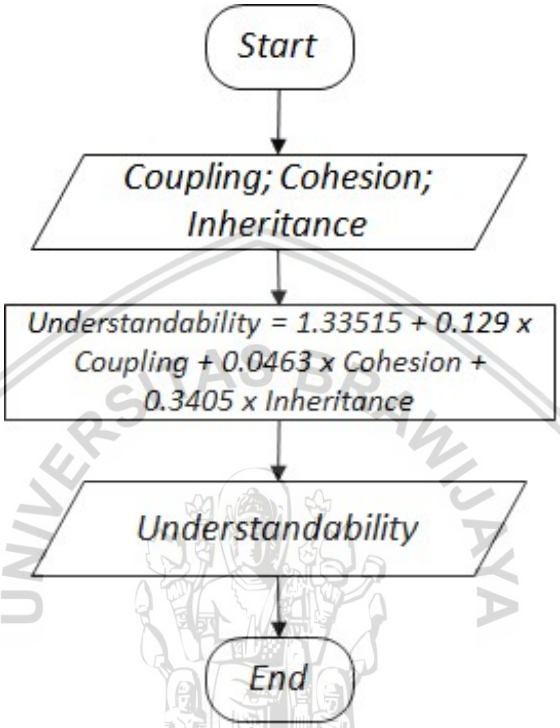
α_0 : Kovarian *understandability* bernilai 1.33515

β_1 : Koefisien *coupling* bernilai 0.129

β_2 : Koefisien *cohesion* bernilai 0.0463

β_3 : Koefisien *inheritance* bernilai 0.3405

Dikarenakan metrik ini berbentuk regresi linear multivariabel maka persebaran nilai yang dihasilkan bersifat monoatomik. Semakin tinggi nilai *understandability* yang dihasilkan maka semakin tinggi usaha untuk memahami diperlukan. Pada Gambar 2.5 mengilustrasikan bagaimana proses perhitungan *understandability* menggunakan *multivariate undestandability metric*:



Gambar 2.5 Algoritme *Multivariate Undestandability Metric*

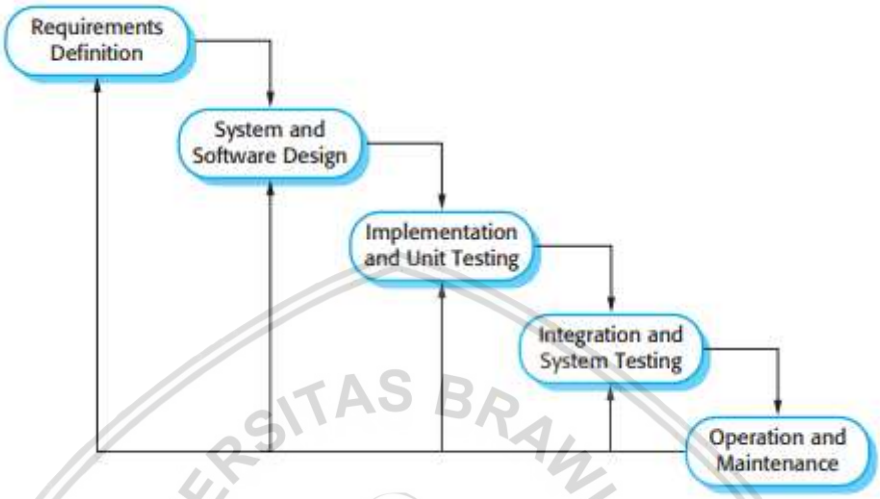
2.7. Software Process Model

Model proses perangkat lunak adalah sebuah representasi penyerderhanaan dari sebuah proses perangkat lunak. Setiap model proses mewakili sebuah proses dari perspektif tertentu dengan demikian hanya menyediakan sebagian informasi mengenai proses tersebut. Model proses ini berbentuk umum dan tidak menjelaskan secara definitif sebuah proses perangkat lunak, melainkan sebuah abstraksi dari proses yang dapat digunakan untuk menjelaskan pendekatan yang berbeda untuk pengembangan perangkat lunak. Model proses perangkat lunak dapat dianggap sebagai kerangka kerja yang dapat diperluas dan disesuaikan untuk menciptakan proses rekayasa perangkat lunak yang lebih spesifik (Sommerville, 2011).

Secara garis besar proses perangkat lunak dianggap sebagai siklus hidup dalam kerangka waktu yang membentang dari pengembangan sistem baru dimulai dari munculnya ide, melakukan implementasi, menjaga kelangsungan hidup, sampai akhirnya sistem tersebut tidak bekerja lagi. Model proses perangkat lunak memiliki beberpa jenis pendekatan yang disebut sebagai *software development life cycle (SDLC)*. Beberapa yang sering digunakan adalah *waterfall model, V-shape,*

spiral, dan *agile model* (Isaias & Issa, 2015). Adapun dalam penulisan kali ini penulis menggunakan *waterfall model* karena prinsipnya bahwa seluruh kebutuhan telah dipahami dengan baik dan telah baku tanpa ada kemungkinan untuk terjadi perubahan yang radikal selama pengembangan sistem.

2.1.1 Waterfall Model



Gambar 2.6 *Waterfall Model*

Sumber: (Sommerville, 2011)

Model proses pengembangan perangkat lunak yang pertama kali digunakan berasal dari rekayasa sistem yang lebih umum (Royce, 1970). Model ini diilustrasikan pada Gambar 2.1. Dikarenakan bentuknya yang seperti deretan urut dari satu fase ke fase yang lain menyerupai air terjun maka model ini dikenal dengan istilah *waterfall model*. *Waterfall model* secara prinsip merupakan proses berdasarkan perencanaan, kita harus merencanakan dan menjadwalkan segala proses aktifitas sebelum memulai pelaksanaanya dan dokumentasi diproduksi pada setiap fase. Permasalahan utama pada *waterfall model* adalah tidak fleksibelnya untuk membagi proyek kedalam tahap yang berbeda. Komitmen harus dilakukan pada tahap awal dalam proses, yang membuat model ini sulit untuk menanggapi perubahan pada kebutuhan pelanggan. Pada prinsipnya *waterfall model* hanya harus digunakan pada saat seluruh kebutuhan telah dipahami dengan baik dan tidak mungkin untuk terjadi perubahan yang radikal selama pengembangan sistem (Sommerville, 2011).

Tahap-tahap utama dalam *waterfall model* mencerminkan secara langsung kegiatan pengembangan mendasar, diantara lain (Sommerville, 2011):

1. Analisis kebutuhan dan pendefinisian layanan yang dimiliki sistem, batasan, dan tujuan yang ditetapkan berdasarkan konsultasi dengan pengguna sistem. Setelah ditetapkan secara rinci hal-hal tersebut dipersiapkan sebagai spesifikasi sistem.
2. Perancangan sistem dan sistem perangkat lunak yang mengalokasikan baik kebutuhan perangkat keras maupun perangkat lunak dengan mendirikan

arsitektur sistem secara keseluruhan. Perancangan perangkat lunak melibatkan identifikasi dan pengGambaran abstraksi sistem dan hubungan mereka.

3. Implementasi dan pengujian unit, selama tahap ini perancangan perangkat lunak diwujudkan sebagai sebuah rangkaian *program* atau unit-unit *program*. Pengujian unit melibatkan proses verifikasi bahwa setiap unit telah memenuhi spesifikasinya.
4. Pengujian integrasi dan pengujian sistem dilakukan dengan mengintegrasikan unit-unit *program* dan mengujinya sebagai satu sistem yang lengkap untuk memastikan bahwa kebutuhan perangkat lunak telah dipenuhi. Setelah diuji sistem perangkat lunak akan dikirim kepada pelanggan.
5. *Maintenance* (tidak selalu) biasanya merupakan fase terpanjang dalam siklus hidup perangkat lunak yang dilakukan setelah sistem dipasang dan ditempatkan didalam lingkungan pengguna. Pemeliharaan melibatkan pengkoreksian kesalahan yang tidak ditemukan pada tahap awal pada siklus hidup, meningkatkan implementasi dari unit sistem dan mengembangkan layanan yang dimiliki sistem sebagaimana kebutuhan baru ditemukan.

2.8. Unified Modelling Language

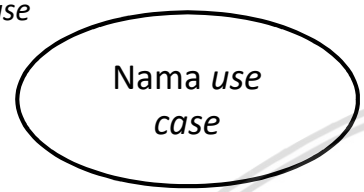
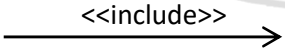
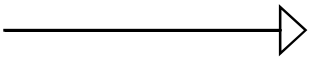
UML adalah kumpulan diagram yang digunakan untuk menentukan berbagai aspek seperti kebutuhan dan desain sistem perangkat lunak. UML digunakan sebagai standar notasi untuk memvisualisasikan dan membangun artefak dari sistem perangkat lunak serta membuat bisnis model termasuk sistem lain non-perangkat lunak. Dalam praktiknya UML telah menjadi standar dalam merancang dan mengembangkan sistem berbasis objek (Hamdy, Elsoud, & El-Halawany, 2011). UML diagram terbagi dalam 2 kategori yaitu *structural diagram* dan *behavioral diagram*. *Structural diagram* merepresentasikan aspek statis dari sistem yaitu bentuk atau struktur dari sistem. *Structural diagram* antara lain yaitu *class diagram*, *object diagram*, *component diagram*, *deployment diagram*. Sedangkan *behavioral diagram* merepresentasikan aspek dinamis dari sistem yaitu perubahan karakteristik atau sifat yang terjadi pada sistem. *Behavioral diagram* antara lain yaitu *use case diagram*, *sequence diagram*, *statechart diagram*, *activity diagram*,*collaboration diagram*. Dalam penulisan kali ini penulis hanya melibatkan 3 macam diagram yaitu *use case diagram*, *sequence diagram*, dan *class diagram* (Alhumaidan, 2012).

2.8.1. Use case Diagram

Use case diagram adalah salah satu diagram UML yang digunakan untuk mewakili fungsi atau kebutuhan sistem. *Use case* menunjukkan fungsi yang akan disediakan sistem dan pengguna yang akan berkomunikasi dengan sistem untuk menggunakan fungsi tersebut (Bennett, McRoob, & Farmer, 2011). *Use case*

diagram meliputi notasi seperti aktor, *use case*, asosiasi dan sistem atau subsistem. *Use case diagram* menunjukan interaksi antara aktor dan sistem untuk mencapai tujuan tertentu. Pengguna atau eksternal sistem eksternal keduanya dianggap sebagai aktor. Sedangkan *use case* mendefinisikan fungsi sistem dari perspektif pengguna dan digunakan untuk mendokumentasikan cakupan sistem (Alhumaidan, 2012). Pada Tabel 2.2 menujukan notasi yang digunakan pada *use case diagram*.

Tabel 2.2 Notasi *Use case Diagram*

Notasi	Deskripsi
Use case 	Fungsionalitas yang disediakan sistem sebagai unit unit yang saling bertukar pesan antar unit atau aktor, biasanya dinyatakan dengan menggunakan kata kerja diawal frase nama *use case.*
Aktor 	Orang, proses, atau sistem lain yang berinteraksi dengan sistem informasi dari luar sistem. Aktor adalah gambar orang, biasanya dinyatakan menggunakan kata benda di awal frase nama aktor.
Asosiasi	Hubungan komunikasi antara aktor dan *use case* mengenai apa yang dapat dilakukan oleh aktor.
Extend 	*Extend* merupakan hubungan pada *use case* yang menunujukkan bahwa ada sebuah *use case* yang melanjutkan perilaku dari *use case* yang lain.
Include 	*Include* merupakan hubungan pada *use case* dimana ada sebuah *use case* yang menggunakan kembali perilaku dari *use case* yang lain.
Generalisasi 	Hubungan generalisasi adalah hununganantara dua buah *use case* dimana fungsi yang satu adalah fungsi yang lebih umum dari lainnya.

Sumber: Diadaptasi dari (Alhumaidan, 2012)

2.8.1.1. *Use Case Scenario*

Use case scenario merupakan penjelasan secara tekstual dari sekumpulan *scenario* interaksi. Setiap *scenario* mendeskripsikan urutan aksi/langkah yang dilakukan aktor ketika berinteraksi dengan sistem, baik yang berhasil maupun yang

gagal. *Use case scenario* harus ditulis dengan rinci dan akurat dengan menghindari penjelasan yang terlalu umum. Berikut ini bagian-bagian penting dalam *use case scenario* (Kurniawan, 2018):

1. Aktor primer, yaitu aktor yang menginisiasi layanan sistem untuk mencapai tujuan dari aktor tersebut. Jumlah aktor primer dimungkinkan lebih dari 1.

2. Prakondisi, yaitu kondisi spesifik yang harus terpenuhi sebelum sebuah UC bisa diinisiasi atau dieksekusi oleh aktor primer. Jumlah prakondisi bisa lebih dari 1 keadaan.

3. Alur utama, yaitu jalur interaksi yang mengarahkan pada skenario yang berhasil sehingga tujuan aktor bisa terpenuhi. Jalur ini hanya terdiri dari 1 jalur saja.

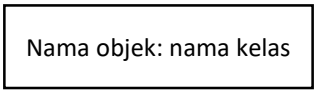
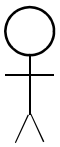
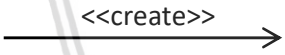
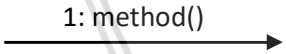
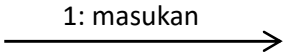
4. alur alternatif, yaitu jalur alternatif dari interaksi yang terjadi antar aktor dengan sistem yang mencakup pencabangan (pilihan) maupun skenario yang gagal sehingga tujuan aktor tidak terpenuhi. Jalur ini bisa terdiri dari lebih dari 1 jalur kemungkinan.

5. kondisi akhir, yaitu kondisi spesifik yang harus terjadi ketika UC berhasil dijalankan atau dieksekusi secara lengkap, sebagai representasi dari tujuan yang ingin dicapai oleh aktor primer. Jumlah kondisi akhir bisa lebih dari 1 keadaan.

2.8.2. *Sequence Diagram*

Sequence diagram adalah *behavioral diagram* yang menunjukkan interaksi antara objek yang disusun dalam urutan waktu. *Sequence diagram* menunjukkan interaksi antara objek-objek di mana mereka dimodelkan sesuai peran mereka dan berkomunikasi melalui pesan yang lewat (Bennett, McRoob, & Farmer, 2011). Pada *sequence diagram* terdapat notasi seperti nama diagram, objek-objek, pesan, aktivasi dan sumbu waktu. *Sequence diagram* diagram digunakan dalam perancangan untuk menangkap dan memodelkan pesan didalam sistem dengan berfokus pada identifikasi perilaku. Secara khusus, *sequence diagram* mewakili alur peristiwa, pesan dan interaksi antara objek dari sistem secara dua dimensi. Interaksi antara objek ditampilkan pada garis horizontal dan waktu ditampilkan pada garis vertikal (Alhumaidan, 2012). Pada Tabel 2.3 menujukan notasi yang digunakan pada *use case diagram*.

Tabel 2.3 Notasi *Sequence Diagram*

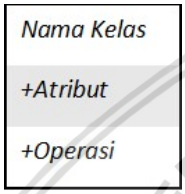
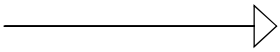
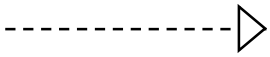
Notasi	Deskripsi
Objek 	Menyatakan objek yang berinteraksi pesan.
Aktor 	Orang, atau sistem lain yang berinteraksi dengan sistem informasi yang akan dibuat di luar sistem informasi. Biasanya dinyatakan menggunakan kata benda diawal frase nama aktor.
Lifeline	Menyatakan kehidupan suatu objek.
Waktu aktif 	Menyatakan objek dalam keadaan aktif dan berinteraksi, semua yang terhubung dengan waktu aktif ini adalah sebuah tanggapan yang dilakukan didalamnya.
Pesan *create* 	Menyatakan suatu objek membuat objek lain, arah panah mengarah pada objek yang dibuat.
Pesan *call method* 	Menyatakan suatu objek memanggil operasi yang ada pada objek lain atau dirinya sendiri. Arah panah mengarah pada objek yang memiliki operasi.
Pesan tipe *send message* 	Menyatakan bahwa suatu objek mengirimkan data/masukan/informasi ke objek lainya. Arah panah mengarah pada objek yang dikirim.
Pesan tipe *return message*	Menyatakan bahwa suatu objek yang telah menjalankan suatu operasi atau metode menghasilkan suatu kembalian ke objek tertentu. Arah panah mengarah pada objek yang menerima kembalian.

Sumber: Diadaptasi dari (Alhumaidan, 2012)

2.8.3. *Class Diagram*

Class diagram adalah *structure diagram* yang digunakan untuk menunjukan kelas dan relasinya terhadap satu sama lain. Pada *class diagram* terdapat notasi seperti kelas, atribut, operasi dan relasi (Bennett, McRoob, & Farmer, 2011). Kelas dalam diagram UML digunakan untuk menangkap informasi tentang sistem yang akan dikembangkan. Kelas merupakan sebauh artefak pada diagram UML yang dapat membuat sejumlah objek yang saling berbagi atribut, operasi, hubungan antara objek, dan beberapa semantik lain didalam diagram (Alhumaidan, 2012). Pada Tabel 2.4 menujukan notasi yang digunakan pada *use case diagram*.

Tabel 2.4 Notasi *Sequence Diagram*

Notasi	Deskripsi
Class 	Kelas pada struktur sistem. Di mana terdapat nama kelas, atribut, operasi, dan hak akses ke setiap atribut dan operasi.
Asosiasi	Relasi antarkelas dengan makna umum yaitu semua objek memiliki daur hidup sendiri dan tidak memiliki pemilik.
Agregasi 	Bentuk spesialisasi dari asosiasi yang memiliki hubungan "*has-a*" yaitu semua objek memiliki daur hidup sendiri akan tetapi memiliki pemilik.
Komposisi 	Bentuk spesialisasi dari agregasi yang memiliki hubungan "*part-of*" yaitu semua objek tidak memiliki daur hidup sendiri dan memiliki pemilik.
Generalisasi 	Relasi antarkelas dengan makna generalisasi spesialisasi (umum-khusus)
Dependency	Relasi antarkelas dengan makna kebergantungan antarkelas.
Realisasi 	Realisasi mirip seperti generaliasi yang mengindikasi terdapat implementasi perilaku dari sebuah kelas pada kelas lain.

Sumber: Diadaptasi dari (Alhumaidan, 2012)

2.9. Pengujian Perangkat Lunak

Pengujian dimaksudkan untuk menunjukkan bahwa suatu program telah memenuhi apa yang dimaksudkan untuk dilakukan dan untuk menemukan cacat dalam program sebelum digunakan. Ketika kita menguji perangkat lunak, kita menjalankan sebuah program menggunakan data buatan. Pemeriksaan hasil uji coba digunakan untuk menemukan kesalahan, anomali, atau informasi tentang atribut non-fungsional program. Proses pengujian memilik dua tujuan yang jelas yaitu (Sommerville, 2011):

1. Untuk menunjukkan kepada pengembang dan pelanggan bahwa perangkat lunak memenuhi seleuruh kebutuhannya. Secara khusus, ini berarti harus ada setidaknya satu tes untuk setiap kebutuhan.
2. Untuk menemukan situasi di mana perilaku perangkat lunak tidak benar, tidak diinginkan, atau tidak sesuai dengan spesifikasinya. Ini adalah konsekuensi dari cacat perangkat lunak.

Secara praktikal, pengujian perangkat lunak mencakup semua kegiatan pengujian yang dilakukan oleh tim yang mengembangkan sistem. Penguji perangkat lunak biasanya *programmer* yang mengembangkan perangkat lunak itu, meskipun ini tidak selalu terjadi. Terkadang dalam melakukan pengujian terdapat *programmer* yang berasosiasi dengan penguji atau bahkan tim penguji dipisah secara independen dari *programmer* atau tim pengembang (Sommerville, 2011). Karena pengujian adalah serangkaian kegiatan yang dapat direncanakan sebelumnya dan dilakukan secara sistematis. Dalam melakukan pengujian perangkat lunak harus mampu mengakomodasi pengujian tingkat rendah yang diperlukan untuk memverifikasi bahwa segmen *source code* telah diterapkan dengan benar serta pengujian tingkat tinggi yang memvalidasi fungsi sistem utama terhadap kebutuhan pengguna. Secara garis besar tahapan dalam pengujian terdiri dari pengujian unit, pengujian integrasi, pengujian validasi dan pengujian sistem (Pressman, 2010). Namun pada penelitian kali ini penulis hanya melakukan tiga tahapan pengujian yaitu pengujian unit, pengujian integrasi dan pengujian validasi.

2.9.1. Pengujian Unit

Pengujian unit dilakukan sebagai bentuk upaya verifikasi pada unit perancangan perangkat lunak terkecil berupa komponen atau modul perangkat lunak. Pengujian unit berfokus pada pemrosesan logika internal dan struktur data dalam batas-batas suatu komponen. Jenis pengujian ini dapat dilakukan secara paralel untuk beberapa komponen. Pendekatan menggunakan *basis path* atau *control structure* adalah strategi yang dapat dilakukan untuk menentukan kasus uji pada pengujian unit. Kasus uji merupakan semua jalur independen yang dilakukan untuk memastikan bahwa semua pernyataan dalam modul telah dijalankan setidaknya sekali. Kasus uji harus dirancang untuk mengungkap kesalahan baik karena kesalahan perhitungan, perbandingan yang salah, atau aliran kontrol yang tidak tepat. Setiap kondisi diuji untuk memastikan bahwa modul beroperasi dengan baik sesuai yang ditetapkan. Dan akhirnya, semua jalur penanganan *error* diuji (Pressman, 2010).

2.9.2. Pengujian Integrasi

Pengujian integrasi adalah proses pengujian di mana individu atau unit perangkat lunak digabungkan dan diuji untuk mengevaluasi interaksi di antara unit tersebut. Dalam pengujian Integrasi, modul atau unit perangkat lunak terintegrasi secara logis dan diuji sebagai suatu kelompok. Pengujian integrasi berfokus pada pemeriksaan komunikasi data di antara unit-unit yang ada. Dalam melakukan pengujian integrasi terdapat beberapa teknik seperti *top-down testing*, *bottom-up testing*, *big bang*, dan *sandwich (hybrid)* (Pressman, 2010). Pada penelitian kali ini penulis menggunakan *top-down testing* untuk pengujian integrasi.

Pengujian integrasi sacara *top-down* adalah teknik pengujian integrasi yang memandang modul bergerak ke bawah melalui hirarki kontrol, dimulai dengan modul kontrol utama ke modul ke arah modul tingkat bawah. Dalam menggunakan teknik ini modul tingkat bawah yang dipanggil oleh modul utama diganti dengan *stub*, yaitu sebuah *dummy module/method* (modul palsu) (Pressman, 2010).

2.9.3. Pengujian Validasi

Pengujian validasi secara sederhana didefinisikan sebagai pengujian untuk menentukan apakah sistem yang dikembangkan telah dapat memenuhi kebutuhan pengguna. Untuk memvalidasi perangkat lunak dilakukan melalui serangkaian tes yang menunjukkan kesesuaian dengan kebutuhan. Pengujian direncanakan dengan menguraikan kelas-kelas tes yang akan dilakukan, dan prosedur uji mendefinisikan kasus uji secara spesifik memastikan bahwa semua kebutuhan fungsional dipenuhi, semua perilaku kebutuhan tercapai, semua konten disajikan secara benar, dan secara dokumentasi terpenuhi (Pressman, 2010).

Dalam melakukan pengujian validasi salah satu cara yang biasa dilakukan adalah dengan menggunakan *equivalence partitioning*. *Equivalence partitioning* merupakan metode yang membagi domain masukan dari suatu sistem ke dalam kelas-kelas/partisi data yang dapat membentuk kasus uji. Kelas data ini biasanya disebut dengan *equivalence class*, yang merepresentasikan keadaan valid dan tidak valid dari masukan sebuah kondisi (Pressman, 2010).

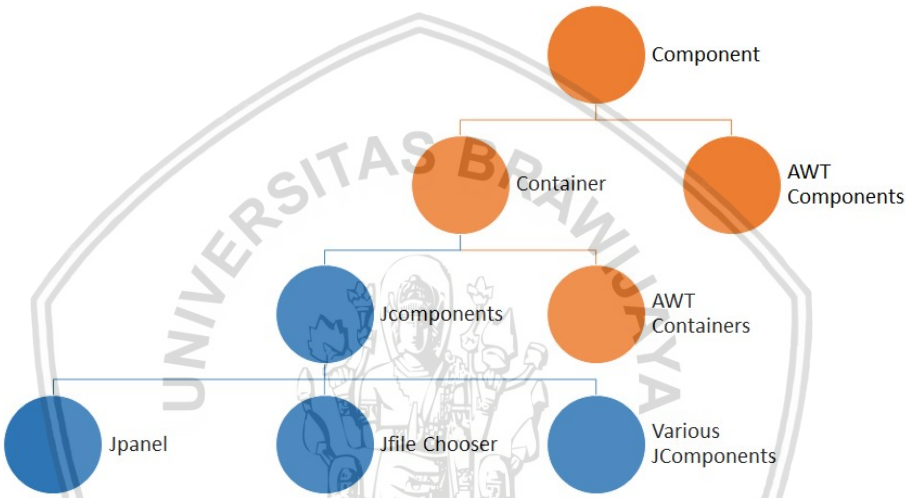
2.10. Teknologi Pengembangan

Pada sub-bab Teknologi Pengembangan akan menjabarkan penggunaan teknologi yang digunakan dalam membangun sistem perangkat lunak dalam penelitian kali ini yang meliputi API/*library* seperti Swing dan JAXP XML Parser.

2.10.1. Swing

Swing digambarkan sebagai satu set komponen grafis yang dapat disesuaikan penampilan dan perilakunya. Swing merupakan generasi baru GUI yang diciptakan oleh Sun Microsystems untuk memungkinkan pengembangan aplikasi berbasis *enterprise* pada Java dan merupakan API GUI utama pada lingkungan pemrograman Java. Swing bukan merupakan singkatan akan tetapi

sebuah nama yang merepresentasikan pilihan kolaboratif para perancangnya ketika projek mereka didepak pada akhir tahun 1996. Swing meskipun dikembangkan terpisah dengan Java Development Kit, namu tetap membutuhkan setidaknya JDK versi 1.1.5 untuk berjalan. Swing memberikan keuntungan seperti *cross-platform compatibility*, perancangannya berorientasi objek, dan memiliki lebih banyak fitur dibanding pendahulunya yaitu AWT (*Abstract Windowing Toolkiit*). Perlu digaris bawahi meskipun Swing memiliki fitur yang lebih banyak dari AWT bukan berarti Swing diciptakan untuk menggantikan penggunaan AWT karena pada dasarnya Swing dibangun diatas AWT dan sangat bergantung pada mekanisme *event-handing* milik AWT. Pada lingkungan pemrograman Java pada Gambar 2.6 berikut menampilkan fitur yang ada pada Swing/AWT dalam hirarki (Cole, dkk. 2002).

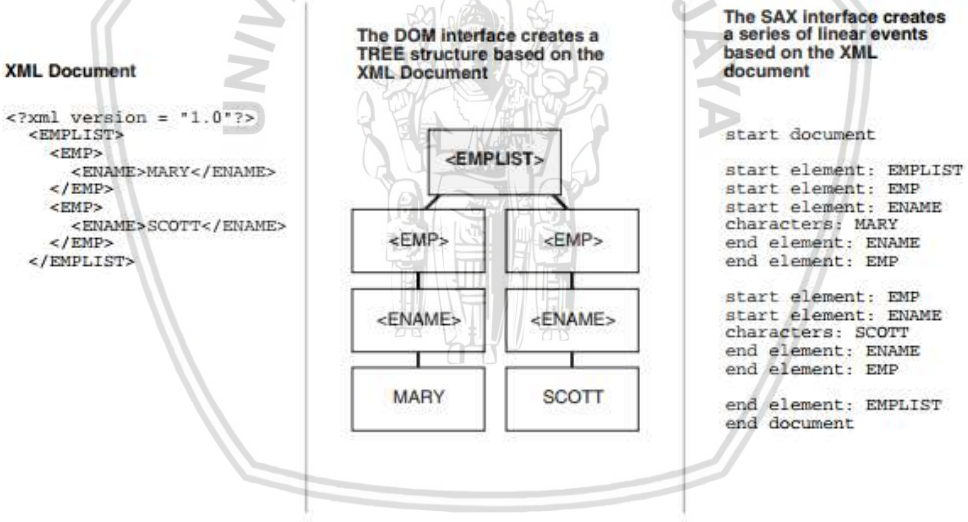


Gambar 2.7 Fitur Swing/AWT

Sumber: (Cole, Loy, dkk, 2002)

2.10.2. *XML* Parser

Sebuah XML *parser* adalah prosesor yang mampu membaca dokumen XML dan menentukan struktur dan properti data. XML *parser* bekerja dengan memecah data kedalam beberpa bagian dan menyediakan data tersebut kepada dokumen lain. XML prosesor dapat mengakses data XML yang telah dipecah secara terprogram menggunakan dua buah API yaitu SAX dan DOM. SAX digunakan jika kita butuh untuk mengakses atau mencari secara serial data pada elemen atau operasi berdasarkan elemen tanpa ada kebutuhan untuk memanipulasi struktur XML. Dengan SAX dokumen XML diuraikan tanpa mengkonsumsi terlalu banyak memori serta lebih cepat dibandingkan DOM. Sedangkan DOM API merepresentasikan dokumen XML kedalam *node-tree* yang dapat dimanipulasikan atau diarahkan. Penggunaan DOM memiliki beberapa keunggulan dibanding SAX seperti penggunaanya yang lebih mudah karena DOM menguraikan XML kedalam struktur *tree* yang familiar, menyediakan pengguna kemampuan untuk menambahkan, menghapus, membaca, dan mengganti nama elemen atau atribut. Pada Gambar 2.7 menampilkan pada bagian kiri terdapat dokumen XML dan perbandingan dari penggunaan DOM pada bagian tengah dan SAX pada bagian kanan. Berikut perbandingannya (Ashdown, Greenberg, & Melnick, 2014):



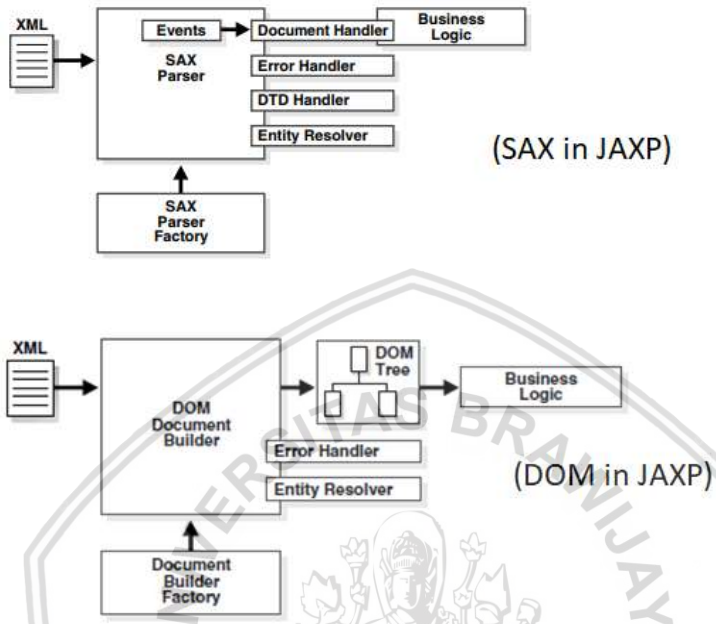
Gambar 2.8 Perbandingan Pengunaan DOM API dan SAX API

Sumber: (Ashdown, Greenberg, & Melnick, 2014)

2.10.2.1. *JAXP XML Parser*

Pada lingkungan pengembangan menggunakan Java untuk menguraikan dokumen XML dikenal sebuah API yaitu JAXP (Java API for XML Processing). JAXP menghasilkan antarmuka dan kelas implementasi yang sesuai dengan skema XML. Dengan menggunakan JAXP pengembang yang menggunakan Java dapat melakukan otomatisasi pemetaan antara dokumen XML dan kode Java, yang memungkinkan program menggunakan kode yang dihasilkan untuk membaca, memanipulasi, dan membuat ulang data XML. JAXP API memungkinkan kita untuk

mengimplementasi DOM atau SAX *parser*. Singkatnya, JAXP memberikan keuntungan untuk pengembangan aplikasi XML pada lingkungan pengembangan menggunakan *Java*. Artsitektur JAXP adalah sebagai berikut pada Gambar 2.8 yang menunjukkan bagaimana penggunaan SAX pada JAXP dan DOM pada JAXP (Ashdown, Greenberg, & Melnick, 2014):



Gambar 2.9 Arsitektur JAXP

Sumber: (Ashdown, Greenberg, & Melnick, 2014)

2.11. Perhitungan Efisiensi

Berdasarkan ISO-9241, efisiensi didefinisikan sebagai jumlah tenaga atau usaha yang dibutuhkan dalam rangka untuk menjamin suatu tujuan dapat tercapai dengan benar. Kunci dalam mengukur jumlah usaha ini biasanya dihitung berdasarkan lama waktu yang dibutuhkan pengguna/partisipan untuk mencapai suatu tujuan secara lengkap. Terdapat dua macam perhitungan efisiensi yaitu *time based efficiency* (efisiensi waktu) dan *overall relative efficiency* (efisiensi secara keseluruhan). *Time based efficiency* digunakan untuk menghitung efisiensi waktu. Nilai didapatkan dengan melihat kecepatan seorang pengguna sistem dalam menyelesaikan sebuah tugas (task) yang diberikan. Sedangkan *overall relative efficiency* digunakan untuk menghitung presentase dari waktu yang dibutuhkan pengguna yang sukses mengerjakan tugas dalam kaitannya dengan total waktu yang dibutuhkan oleh semua pengguna (Seergev, 2010).

Berikut ini merupakan persamaan perhitungan efisiensi secara *time besed efficiency* pada persamaan 2 dan *overall relative efficiency* pada persamaan 3:

$$Time\ Based\ Efficiency = \frac{\sum_{j=1}^{R} \sum_{i=1}^{N} \frac{n_{ij}}{t_{ij}}}{NR}. \qquad (2)$$

$$Overall\ \text{Re}lative\ Effeciency = \frac{\sum_{j=1}^{R}\sum_{i=1}^{N} n_{ij}t_{ij}}{\sum_{j=1}^{R}\sum_{i=1}^{N} t_{ij}} \times 100\% \quad (3)$$

Dimana,

N :Total jumlah tugas

R : Total jumlah pengguna

n_{ij} : Hasil dari tugas i yang berhasil dikerjakan user j (bernilai 1 jika berhasil dan 0 jika gagal).

t_{ij} : Lama waktu yang dibutuhkan pengguna untuk menyelesaikan tugas

2.12. Spearman's Rank Correlation

Koefisien *spearman's rank correlation* merupakan pengukuran statistik yang diperkenalkan oleh Charles Spearman pada tahun 1904 yang digunakan untuk mencari tahu seberapa kuat hubungan monoatomik diantara dua buah data kuantitatif yang pasangkan. *Spearman's rank correlation* dinotasikan sebagai r_s dan dimodelkan untuk memiliki batasan nilai berupa $-1 \leq r_s \leq 1$. Nilai dari *spearman's rank correlation* diinterpretasikan jika nilai r_s semakin mendekati 1 maka hubungan monoatomiknya pasangan data itu semakin kuat. Nilai korelasi secara verbal dapat dideskripsikan sebagai berikut pada Tabel 2.5 (Zar, 2005):

Tabel 2.5 Koefisien Spearman's Rank Correlation

Nilai	Predikat
.00-.19	Sangat lemah
.20-.39	Lemah
.40-.59	Sedang
.60-.79	Kuat
.80-1.0	Sangat Kuat

Berikut ini merupakan perhitungan *spearman's rank correlation* yang ditunjukan pada persamaan 4:

$$r_s = 1 - \frac{6\sum d^2}{n(n^2-1)} \quad (4)$$

Dimana,

r_s : Koefisien *spearman's rank correlation*

d : Selisi nilai dari data yang dibandingkan

n : Jumlah data

BAB 3 METODE PENELITIAN

Pada bab 3 akan membahas metode yang akan digunakan dalam penelitian ini. Langkah-langkah yang dilakukan antara lain terdiri dari studi kepustakaan, analisis kebutuhan sistem, perancangan sistem dan implementasi, pengujian, kemudian pengambilan kesimpulan dan saran. Dalam penelitian ini penulis juga mengembangkan sebuah perangkat lunak yang metode pengembangannya mengadopsi *waterfall model*. Hal ini dikarenakan kebutuhan telah dipahami dengan baik dan tidak mungkin untuk terjadi perubahan yang radikal selama pengembangan sistem. Berdasarkan model tersebut. Pada Gambar 3.1 berikut menunjukkan alur metode penulisan secara umum.

Gambar 3. 1 Alur Metode Penulisan

3.1. Studi Kepustakaan

Studi kepustakaan dilakukan untuk mencari dan mempelajari landasan teori untuk menguatkan penulisan seputar topik yang berkaitan dengan metode pengembangan perangkat lunak, *maintenance* perangkat lunak, urgensi, dan tantangnya, karakteristik pengembangan berbasi objek yang menekan pada *cohesion*, *coupling*, dan *inheritance, OOD metric*, JAXP DOM *parser*, *understandability*, dan macam-macam *metric understandability*.

3.2. Rekayasa Kebutuhan

Mendefinisikan kebutuhan fungsional dan non-fungsional yang akan dimiliki sistem dengan cara mengidentifikasi masalah terlebih dahulu,

mengumpulkan, dan mengkaji pustaka yang menguatkan fakta tentang kebutuhan yang harus dipenuhi sistem dalam mengatasi pengukuran *understandability* sebuah rancangan perangkat lunak. Rekayasa kebutuhan dilakukan menggunakan pendekatan berbasis objek dengan teknik elisitasi kebutuhan menggunakan persona. Tidak hanya itu, dalam rekayasa kebutuhan penulis juga akan mengidentifikasi aktor sistem dan menggali kebutuhan sistem yang terkait dengan lingkungan pengembangan, lingkungan operasi, identifikasi pengguna dan bagaimana mekanisme komunikasi sistem dalam menjalankan tugasnya.

3.3. Perancangan Sistem dan Implementasi

Setelah mendapatkan kebutuhan fungsional dan non-fungsional secara jelas penulis akan melakukan perancangan sistem yang berkaitan dengan perancangan arsitektur, perancangan, komponen, perancangan algoritme, dan perancangan antarmuka pengguna yang dilakukan dengan pendekatan berbasis objek dengan pemodelan perancangan menggunakan *sequence diagram* dan *class diagram*. Selanjutnya untuk melakukan implementasi dari perancangan sistem, sistem dibuat menggunakan bahasa *Java*. Dimana sistem yang dibangun akan berjalan sebagai aplikasi *desktop* dan menerapkan salah satu XML *parser* yang bekerja pada lingkungan *Java* yaitu JAXP untuk melakukan *parsing* berkas XML yang menjadi inputan pada penelitian kali ini. Sistem yang menerima inputan akan menjalankan perhitungan *understandability* yang sesuai dengan persamaan *multivatiare understandabilty metric*. Metrik lain yang digunakan adalah NA untuk mengukur *cohesion*, NAssoc untuk mengukur *coupling*, dan NOC untuk mengkur *inheritance*. Sedangkan untuk implementasi antarmuka pengguna digunakan Swing.

3.4. Pengujian dan Analisis Hasil

Tahap pengujian pada penelitian ini meliputi pengujian unit dengan menggunakan teknik *basis path*, pengujian integrasi dengan menggunakan pendekatan *top-down testing*, dan pengujian validasi menggunakan teknik *equivalence partitioning*. Pengujian efisiensi dilakukan dengan mengukur efisiensi berdarkan *time based efficiency* dan *overall relative efficiency* Kemudian melakukan analisis hasil sistem untuk melihat seberapa kuat hubungan hasil *understandability* yang didapat oleh sistem terhadap usaha melakukan *maintenance* nantinya.

3.5. Kesimpulan dan Saran

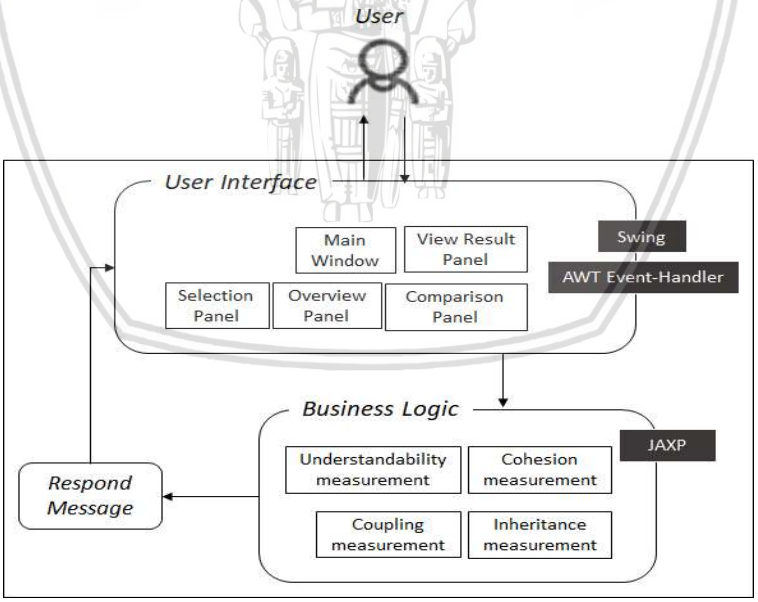
Setelah penelitian selesai ditariklah kesimpulan untuk menjawab rumusan masalah yang telah dijabarkan pada awal penulisan ini berdasarkan pengimplementasian yang sudah dilakukan. Dan pemberian saran yang bertujuan untuk memperbaiki kekurangan yang terdapat pada penelitian ini agar penelitian ini dapat dikembangkan lagi dikemudian hari.

BAB 4 REKAYASA KEBUTHAN

4.1. Deskripsi Umum Sistem

Sistem yang dikembangkan diberi sebutan SUMIT atau *Software Understandability Measurement Instant Tool* yang merupakan sebuah kakas bantu untuk mengukur tingkat *understandability* sebuah rancangan perangkat lunak berupa *class diagram* dengan menggunakan *multivariate understandability metric*. Di dalam *multivariate understandability metric* terdapat 3 variabel bebas yang digunakan yaitu *coupling, cohesion,* dan *inheritance* yang digunakan sebagai penentu nilai *understandability*. *Coupling* akan dihitung berdasarkan jumlah asosiasi pada rancangan sistem (menggunakan metrik NAssoc), *cohesion* berdasarkan jumlah atribut dari kesuluruhan kelas yang ada (menggunakan metrik NA), dan *inheritance* berdasarkan jumlah kelas *child* pada sistem (menggunakan metrik NOC).

Sistem merupakan sebuah aplikasi *desktop* dimana pengguna/user untuk melakukan pengukuran *understandability* harus terlebih dahulu telah menyimpan berkas rancangan sistem berupa *class diagram* kedalm dokumen XML. Berkas rancangan dalam bentuk XML ini akan menjadi masukan bagi sistem. Setelah berkas dimasukkan, sistem akan mengambil variabel-variabel yang dibutuhkan untuk pengukuran *understandability*. Untuk melakukan hal tersebut sistem menggunakan JAXP sebagai XML *processor*.



Gambar 4.1 Arsitektur Sistem

Secara umum sistem berjalan bedasarkan ilustrasi pada Gambar 4.1 dimana *user* dan sistem saling berkomunikasi dengan sistem baik memberi masukan atau pun menerima keluaran melalui *user interface* yang dikelolah oleh Swing dengan mengguna AWT sebagai *event-handler*. Masukan dari *user* yang dikelolah *business logic* sistem jika terdapat berkas XML maka akan diproses oleh

JAXP yang hasilnya kemudian digunakan untuk perhitungan *understandability*. Setelah data masukan dikelolah maka sistem akan memberi *respond message* yang kembali ditampilkan pada *user interface* pengguna yang dikelolah oleh Swing.

4.2. Elisitasi Kebutuhan

Bagian ini memberitahukan kebutuhan yang harus dipenuhi oleh sistem dari permaslahan yang ditemukan pada sisi pengguna perangkat lunak. Elisitasi kebutuhan dilakukan dengan mengkaji kepustakaan terkait pengukuran *understandability.* Dari hasil pengkajian kepustakaan kemudian dibuatlah sebuah persona yang merepresentasikan pengguna. Persona diasumsikan sebagai seorang/tim pengembang perangkat lunak yang ingin mengukur tingkat *understandability* dari rancangan sistem yang telah dibuat.

Dari hasil pembuatan persona adapun definisi kebutuhan ditemukan yang dimiliki oleh perangkat lunak SUMIT adalah:

1. Memilih berkas XML rancangan perangkat lunak.
2. Mengukur porsi *coupling* rancangan perangkat lunak.
3. Mengukur porsi *cohesion* rancangan perangkat lunak.
4. Mengukur porsi *inheritance* rancangan perangkat lunak.
5. Mengukur tingkat *understandability* rancangan perangkat lunak.
6. Melihat hasil pengukuran *understandability* rancangan perangkat lunak.
7. Menampilkan ikhtisar hasil pengkuran rancangan perangkat lunak.
8. Membandingkan tingkat *understandability* antar dua rancangan perangkat lunak.

4.3. Identifikasi Karakteristik Pengguna

Pengguna dari sistem ini adalah individu yang bekerja pada atau telah familiar dengan dunia pengembangan perangkat lunak sehingga memahami istilah-istilah yang digunakan dalam sistem seperti *design, model, class diagram, coupling, cohesion, inheritance,* dan *understandability*. Lebih spesifik, pengguna dari sistem ini adalah mereka para *desainer* perangkat lunak.

Sistem ini memiliki pengguna tunggal yaitu seorang *desainer* perangkat lunak yang menggunakan layanan SUMIT untuk mengkur apakah rancangan sistem yang dibuat sudah memiliki tingkat *understandability* yang baik atau tidak. Dikarenakan hanya ada 1 jenis pengguna yang terlibat dalam sistem ini yaitu desainer penulis akan menggunakan istilah ‘pengguna’ sebagai acuan untuk menyebut *desainer* sebagai pengguna.

Dengan menggunakan sistem ini pengguna akan dapat melakukan hal-hal sebagi berikut yang dijabarkan pada Tabel 4.1:

Tabel 4.1 Karakteristik Pengguna

No	Identifikasi Pengguna	Karakteristik
1.	Pengguna	Pengguna dapat melakukan :

		1. Memilih berkas XML rancangan perangkat lunak. 2. Mengukur porsi *coupling* rancangan perangkat lunak. 3. Mengukur porsi *cohesion* rancangan perangkat lunak. 4. Mengukur porsi *inhertance* rancangan perangkat lunak. 5. Mengukur tingkat *understandability* perancangan perangkat lunak. 6. Melihat hasil pengukuran rancangan perangkat lunak. 7. Menampilkan ikhtisar hasil pengkuran rancangan perangkat lunak. 8. Membandingkan tingkat *understandability* antar dua rancangan perangkat lunak.

4.4. Perhitungan Manual

Perhitungan manual dilakukan untuk memberikan pemahamn mengenai bagaimana perhitungan *understandability* menggunakan *multivariate understandability metric*. Untuk menghitung *multivariate understandability metric* menggunakan 3 jenis variabel bebas yaitu *coupling, cohesoni,* dan *inhetitance*. Dipilihnya ketiga jenis variabel ini karena diduga yang berpengaruh terhadap tingkat *understandability* sebuah rancangan perangkat lunak. Model yang digunakan adalah sebagai berikut:

$$Understandability = 1.33515 + 0.129 * NAssoc + 0.0463 * NA + 0.3405 * NOC$$

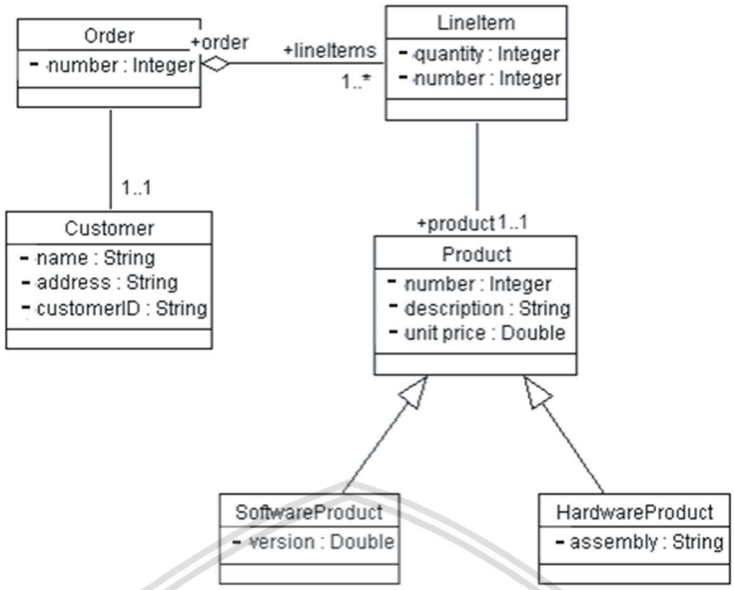
Dimana,

Nassoc : Jumlah asosiasi dalam sistem (untuk mengukur *coupling*)

NA : Jumlah atribut dalam sistem (untuk mengukur *cohesion*)

NOC : Jumlah total dari *child class* (untuk mengukur *inheritance*)

Berikut sampel class diagram dari “sistem A” yang digunakan untuk pengukuran:



Gambar 4.2 Class Diagram Sistem A

Dari sistem A kita mendapatkan data sebagai berikut ditunjukan pada Tabel 4.2:

Tabel 4.2 Data set sistem A

Sistem A		
Coupling	***Cohesion***	***Inheritance***
2	11	2

Dengan menggunakan data set dari tabel 4.2 dan menggunakan persamaan *multivariate understandability metric* diperoleh nilai *understandability* untuk sistem A sebagai berikut:

$$Undestandability = 1.33515 + 0.129 * 2 + 0.0463 * 11 + 0.3405 * 2$$

$$Understanability = 1.33515 + 0.285 + 0.5093 + 0.681$$

$$Understanability = 2.81045$$

4.5. Spesifikasi Kebutuhan Sistem

Pada bagian ini penulis menjabarkan spesifikasi kebutuhan hasil rekayasa kebutuhan yang dibagi kedalam dua macam kebutuhan yaitu kebutuhan fungsional dan kebutuhan non-fungsional.

4.5.1. Kebutuhan Fungsional

Kebutuhan fungsional adalah kumpulan pernyataan mengenai layanan yang harus diberikan/dipenuhi sistem, bagaimana sistem bereaksi kepada sebuah masukan dan bagaimana sistem berperilaku terhadap kondisi tertentu. Pada

sistem yang dikembangkan ini kebutuhan fungsional adalah sebagai berikut dapat dilihat pada Tabel 4.3:

Tabel 4.3 Spesifikasi Kebutuhan Fungsional Sistem

Kode SRS	Kebutuhan	*Use case*
SRS_F_SUMIT_001	Sistem harus dapat melayani keperluan pengguna untuk memilih dan memasukan berkas rancangan perangkat lunak.	*Browse* Berkas
	1.1. Sistem menyediakan sebuah tombol *browse* yang dilengkapi dengan textfield yang akan terisi otomatis dengan nama berkas yang dipilih pengguna jika sudah ada berkas yang terpilih. 1.2. Sistem memberikan sebuah kotak dialog untuk melakukan *browse* berkas pada direktori lokal milik pengguna. 1.3. Secara *default* berkas yang ditampilkan berupa direktori dan berkas dengan format XML. 1.4. Sistem hanya dapat menerima berkas dalam format XML yang menggunkan struktur *simple* milik Visual Paradigm sebagai inputan.	
SRS_F_SUMIT_002	Sistem harus dapat memberikan layanan dalam mengukur nilai porsi *coupling* rancangan perangkat lunak sesuai berkas XML rancangan perangkat lunak yang dimasukan pengguna.	Ukur Porsi *Coupling*
	2.1. Sistem mengukur *coupling* rancangan perangkat lunak menggunakan metrik NAssoc yaitu metrik yang mengukur nilai *coupling* berdasarkan jumlah asosiasi yang dimiliki sebuah rancangan perangkat lunak. 2.2. Sistem mengalikan hasil *coupling* dari metrik NAssoc dengan koefisien *coupling* pada *multivariate understandability metric.* 2.3. Nilai minimum pengukuran *coupling* dengan menggunakan metrik *NAssoc* bernilai 0.	

	2.4. Sistem memberikan sebuah tombol untuk memulai proses perhitungan. 2.5. Terdapat *progress bar* yang mengisyaratkan berjalanya proses perhitungan. 2.6. Perhitungan tidak akan dilakukan jika tidak ada target berkas rancangan perangkat lunak yang dipilih.	
SRS_F_SUMIT_003	Sistem harus dapat memberikan layanan dalam mengukur nilai porsi *cohesion* rancangan perangkat lunak sesuai berkas XML rancangan perangkat lunak yang dimasukan pengguna.	Ukur Porsi *Cohesion*
	3.1. Sistem mengukur *cohesion* rancangan perangkat lunak menggunakan metrik NA yaitu metrik yang mengukur nilai *cohesion* berdasarkan jumlah atribut yang dimiliki sebuah rancangan perangkat lunak. 3.2. Sistem mengalikan hasil *cohesion* dari metrik NA dengan koefisien *cohesion* pada *multivariate understandability metric*. 3.3. Nilai minimum pengukuran *cohesion* dengan menggunakan metrik NA bernilai 0. 3.4. Sistem memberikan sebuah tombol untuk memulai proses perhitungan. 3.5. Terdapat *progress bar* berjalannya yang mengisyaratkan proses perhitungan. 3.6. Perhitungan tidak akan dilakukan jika tidak ada target berkas rancangan perangkat lunak yang dipilih.	
SRS_F_SUMIT_004	Sistem harus dapat memberikan layanan dalam mengukur nilai porsi *inheritance* rancangan perangkat lunak sesuai berkas XML rancangan perangkat lunak yang dimasukan pengguna.	Ukur Porsi *Inheritance*
	4.1. Sistem mengukur *inheritance* rancangan perangkat lunak menggunakan metrik NOC yaitu metrik	

	yang mengukur nilai *inheritance* berdasarkan jumlah kelas *child* yang dimiliki sebuah rancangan perangkat lunak. 4.2. Sistem mengalikan hasil *inheritance* dari metrik NOC dengan koefisien *inheritance* pada *multivariate understandability metric*. 4.3. Nilai minimum pengukuran *inheritance* dengan menggunakan metrik NOC bernilai 0. 4.4. Sistem memberikan sebuah tombol untuk memulai proses perhitungan. 4.5. Terdapat *progress bar* yang mengisyaratkan berjalannya proses perhitungan. 4.6. Perhitungan tidak akan dilakukan jika tidak ada target berkas rancangan perangkat lunak yang dipilih.	
SRS_F_SUMIT_005	Sistem harus dapat memberikan layanan dalam mengukur tingkat *understandability* rancangan perangkat lunak sesuai berkas XML rancangan perangkat lunak yang dimasukan pengguna.	Ukur *Understandability*
	5.1. Sistem mengukur *understandability* rancangan perangkat lunak menggunakan multivariate understandability metricyaitu model regresi linear multivariabel yang mengukur nilai *understandability* berdasarkan nilai dari *coupling*, *cohesion*, dan *inheritance*. 5.2. Penggunaan *multivariate understandability metric* akan memberikan nilai *minimum understandability* sebesar 0. 5.4. Sistem memberikan sebuah tombol untuk memulai proses perhitungan.	

	5.5. Terdapat *progress bar* yang mengisyaratkan berjalannya proses perhitungan. 5.6. Perhitungan tidak akan dilakukan jika tidak ada target berkas rancangan perangkat lunak yang dipilih.	
SRS_F_SUMIT_006	Sistem harus menyediakan layanan untuk menampilkan hasil perhitungan *understandability,* porsi *coupling,* porsi *cohesion, dan* porsi *inheritance*.	Lihat Hasil
	6.1. Sistem menyediakan halaman khusus untuk melihat hasil perhitungan *understandability,* porsi *coupling,* porsi *cohesion, dan* porsi *inheritance*. 6.2. Sistem menyediakan sebuah tombol navigasi untuk melihat hasil perhitungan *understandability,* porsi *coupling,* porsi *cohesion, dan* porsi *inheritance*. 6.3. Sistem menampilkan tingkat *understandability* beserta rincian perhitungannya. 6.4. Sistem menampilkan hasil perhitungan porsi *coupling* berserta rincian perhitungnya. 6.5. Sistem menampilkan hasil perhitungan porsi *cohesion* berserta rincian perhitungnya. 6.6. Sistem menampilkan hasil perhitungan porsi *inheritance* berserta rincian perhitungnya.	
SRS_F_SUMIT_007	Sistem harus menyediakan layanan kepada pengguna untuk melihat ikhtisar hasil terkait pengukuran *understandability* rancangan perangkat lunak.	Lihat Ikhtisar
	7.1. Sistem menyediakan halaman khusus untuk menampilkan ikhtisar hasil terkait pengukuran *understandability*. 7.2. Sistem menyediakan sebuah tombol khusus untuk melakukan navigasi ke halaman lihat ikhtisar.	

	7.3. Ikhtisar disajikan dalam bentuk panel-panel. 7.4. Panel pertama berisikan nilai understandability rancangan perangkat lunak berserta masing-masing porsi dari coupling, cohesion, dan inheritance. 7.5. Panel kedua berisikan nilai dari metrik NAssoc, metrik NA, dan metrik NOC. 7.6. Panel ketiga berisikan informasi mengenai total kelas, total atrubut, total operasi, total asosiasi, dan total generalisasi yang ada dalam rancangan perangkat lunak.	
SRS_F_SUMIT_008	Sistem harus dapat menyediakan layanan kepada pengguna untuk membandingkan tingkat *understandability* diantara dua buah rancangan perangkat lunak.	Banding *Design*
	8.1. Sistem menyediakan halaman khusus untuk membandingkat tingkat *understandability* antara dua buah rancangan perangkat lunak. 8.2. Sistem menyediakan sebuah tombol khusus untuk melakukan navigasi ke halaman banding *design*. 8.3. Terdapat *progress bar* yang mengisyaratkan berjalannya proses perbandingan. 8.4. Proses banding hanya dapat dilakukan jika sudah ada sebuah berkas perangkat lunak yang dihitung *understandability*-nya terlebih dahulu dan berkas bandingan berhasil dipilih. 8.5. Sistem menyediakan sebuah tombol *browse* yang dilengkapi dengan textfield yang akan terisi otomatis dengan nama berkas yang dipilih pengguna jika sudah ada berkas bandingan yang terpilih. 8.6. Sistem memberikan sebuah kotak dialog untuk melakukan *browse* berkas	

	bandingan pada direktori lokal milik pengguna. 8.7. Secara *default* berkas yang ditampilkan berupa direktori dan berkas dengan format XML. 8.8. Sistem hanya dapat menerima berkas dalam format XML yang menggunkan struktur *simple* milik Visual Paradigm sebagai inputan.	

4.5.2. Kebutuhan Non Fungsional

Kebutuhan non-fungsional adalah kumpulan pernyataan mengenai batasan yang mempengaruhi berjalannya fungsi utama sistem dengan baik, bagaimana hasil yang dikeluar sistem memenuhi sebuah aspek penilaian tertentu. Pada sistem yang dikembangkan ini kebutuhan fungsional terdiri sebagai berikut pada Tabel 4.4:

Tabel 4.4 Spesifikasi Kebutuhan Non-Fungsional Sistem

Kode SRS	Parameter	Deskripsi Kebutuhan
SRS_NF_SUMIT_001	Efisiensi	Sistem memiliki tingkat efisiensi lebih dari 70% dari segi keberhasilan menjalankan sebuah tugas dan secara waktu lebih cepat daripada perhitungan secara manual pada rancangan yang kompleks.

4.6. Kebutuhan Antarmuka Eksternal

Berikut adalah antarmuka eksternal yang digunakan dalam menunjang pengoperasian sistem. Antarmuka eksternal sistem ini meliputi antarmuka pengguna, antarmuka perangkat keras, antarmuka perangkat lunak.

4.6.1. Antarmuka Pengguna

Pengguna berinteraksi dengan sistem melalui antarmuka aplikasi *desktop*. Antarmuka yang digunakan untuk interaksi antara sistem dengan pengguna dibagi kadalam 5 halaman yaitu halaman utama sebagai *window* utama dan tempat melakukan navigasi ke halaman lainnya, halaman seleksi berkas dan pengukuran untuk melakukan *browse* berkas rancangan perangkat lunak sekaligus untuk memulai pengukuran, halaman untuk melihat hasil pengukuran, halaman membandingakan rancangan perangkat lunak dan halaman untuk melihat ikhitisar hasil pengukuran *understandability*.

Pengguna melakukan akses fungsi-fungsi pada sistem dengan menggunakan *mouse* atau *touchpad*, dan *keyboard* sebagai fungsi inputan ke

sistem. *Output* dari sistem disajikan disetiap halaman SUMIT yang tampil pada monitor.

4.6.2. Antarmuka Perangkat Keras

Perangkat keras yang digunakan pengguna memiliki spesifikasi kebutuhan sebagai berikut:

1. RAM yang digunakan 4GB atau lebih.
2. Monitor dengan resolusi 800x600 pixel atau lebih.
3. *Keyboard* dan *mouse/touchpad* sebagai perangkat untuk interaksi antara pengguna terhadap sistem.

4.6.3. Antarmuka Perangkat Lunak

Antarmuka perangkat lunak dalam sistem memerlukan JAXP API untuk melakukan *parsing* berkas XML rancangan perangkat lunak dan *library* Swing untuk membangun GUI sistem.

4.7. Batasan Sistem

Berikut adalah daftar batasan dari sistem yang dikembangkan pada penulisan ini:

1. Sistem berjalan sebagai aplikasi *desktop*.
2. Berkas yang menjadi inputan pada sistem menggunakan format XML dengan struktur *simple* dari Visual Paradigm.
3. Sistem menggunakan metrik NAssoc untuk mengukur nilai *coupling* berdasarkan jumlah asosiasi yang dimiliki sebuah rancangan perangkat lunak.
4. Sistem menggunakan metrik NA untuk nilai *cohesion* berdasarkan jumlah atribut dalam setiap kelas yang dimiliki sebuah rancangan perangkat lunak.
5. Sistem menggunakan metrik NOC untuk mengukur nilai *inheritance* berdasarkan nilai total keberadaan *child class*.

4.8. Lingkungan Pengembangan

Adapun spesifikasi lingkungan pengembangan sistem sebagai berikut:

1. Menggunakan IDE NetBeans versi 8.2.
2. Bahasa pemrograman yang digunakan adalah bahasa *Java*.
3. Menggunakan JAXP sebagai XML .
4. OS menggunakan Microsoft Windows 7, 8.1, atau 10.
5. Menggunakan Astah Community sebagai modeling tool dan Draw.io untuk membuat *mock-up* antarmuka pengguna.

4.9. Lingkungan Operasi

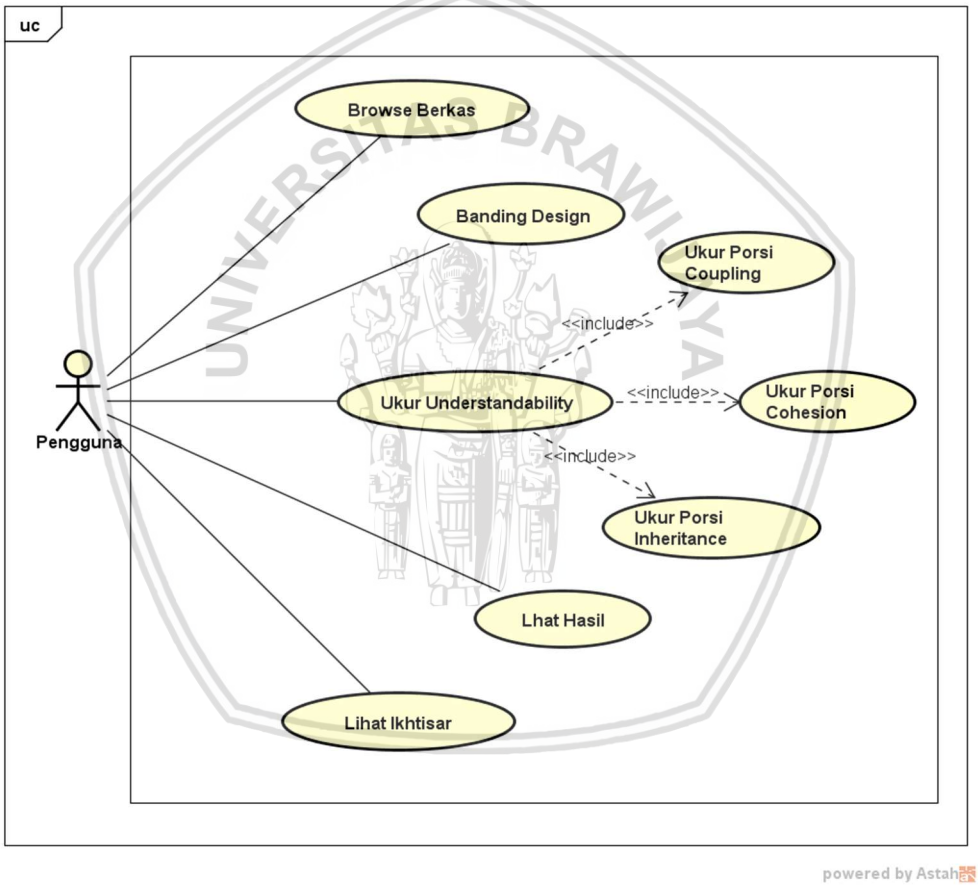
Adapun spesifikasi lingkungan operasi sistem sebagai berikut:

1. OS menggunakan Microsoft Windows 7 atau versi yang lebih tinggi.
2. RAM lebih besar sama dengan 4GB.

4.10. Pemodelan Kebutuhan

Pada pemodelan kebuthan penulis memetakan kebutuhan fungsional sistem yang telah dispesifikasikan kedalam *use case* diagram dan penjelasan rinci mengenai perilaku setiap *use case* akan dijelaskan ke dalam *use case scenario*.

4.10.1. Pemodelan *Use Case Diagram*



Gambar 4.3 *Use Case Diagram* SUMIT

Pada Gambar 4.3 menunjukan Pengguna sebagai aktor tunggal dalam sistem SUMIT yang memiliki kewenangan untuk menggunakan seluruh kebutuhan sistem SUMIT. Kebutuhan sistem SUMIT sesuai dengan yang telah dispesifikasikan berjumlah 8 buah yaitu;

1. *Browse* berkas (SRS_F_SUMIT_001),
2. Ukur Porsi *Coupling* (SRS_F_SUMIT_002),
3. Ukur Porsi *Cohesion* (SRS_F_SUMIT_003),
4. Ukur Porsi *Inheritance* (SRS_F_SUMIT_004),
5. Ukur *Understandability* (SRS_F_SUMIT_005),
6. Lihat Hasil (SRS_F_SUMIT_006),
7. Lihat Ikhtisar (SRS_F_SUMIT_007),
8. dan Banding *Design* (SRS_F_SUMIT_008).

4.10.2. *Use Case Scenario*

Use case *scenario* akan menjelaskan lebih rinci seperti apa setiap kebutuhan pada SUMIT akan berperilaku ketika berkomunikasi dengan Pengguna. Berikut *use case scenario* dari SUMIT:

1. *Use case scenario Browse* Berkas (SRS_F_SUMIT_001)

   Pada Tabel 4.5 menjelaskan secara rinci perilaku dari *use case Browse* Berkas ketika berinteraksi dengan Pengguna.

Tabel 4.5 *Use Case Scenario Browse* Berkas

***Browse* Berkas**	
Kode Kebutuhan	SRS_F_SUMIT_001
Objective	Memberikan Pengguna layanan untuk memilih berkas rancangan perangkat lunak dari direktori lokal.
Aktor	Pengguna
Pre Conditions	1. Pengguna berada pada halaman *browse* berkas.
Main Flow	1. Pengguna menekan tombol *browse*. 2. Sistem menampilkan *dialog file chooser* dengan *default* menampilkan berkas XML. 3. Pengguna memilih berkas yang diinginkan. 4. Pengguna menekan tombol *Open*.
Alternative Flow	1. Jika Pengguna membatalkan pemilihan berkas atau menekan tombol *cancel* maka tidak ada berkas yang terpilih.
Post Conditions	1. Sistem menampilkan *path* berkas yang telah dipilih.

2. *Use case scenario* Ukur Porsi *Coupling* (SRS_F_SUMIT_002)

   Pada Tabel 4.6 menjelaskan secara rinci perilaku dari *use case* Ukur Porsi *Coupling* ketika berinteraksi dengan Pengguna.

Tabel 4.6 *Use Case Scenario* Ukur Porsi *Coupling*

Ukur Porsi *Coupling*	
Kode Kebutuhan	SRS_F_SUMIT_002
Objective	Memberikan Pengguna layanan untuk mengukur porsi *coupling* dari ancangan perangkat lunak yang dipilih.
Aktor	Pengguna
Pre Conditions	1. Pengguna berada pada halaman *browse file*. 2. Pengguna telah memilih sebuah berkas untuk diukur.
Main Flow	1. Pengguna menekan tombol *process*. 2. Sistem melakukan *parsing* berkas XML rancangan perangkat lunak. 3. Sistem menghitung *coupling*. 4. Sistem menghitung nilai porsi *coupling*. 5. Progress bar menampilkan kemajuan proses perhitungan.
Alternative Flow	1. Jika berkas yang dipilih Pengguna tidak ber*format* XML maka sistem menampilkan pemberitahuan kesalahan berkas tidak ber*format* XML. 2. Jika berkas XML tidak memiliki struktur *simple* milik Visual Paradigm maka sistem akan menampilkan pemberitahuan berkas XML tidak sesuai dengan struktur yang bisa di*parsing* SUMIT. 3. Jika tidak ada berkas yang terpilih maka sistem menampilkan pemberitahuan tidak ada berkas yang dipilih Pengguna.
Post Conditions	Sistem berhasil mengukur nilai porsi *coupling* dan nilai *understandability*.

3. *Use case scenario* Ukur Porsi *Cohesion* (SRS_F_SUMIT_003)

   Pada Tabel 4.7 menjelaskan secara rinci perilaku dari *use case* Ukur Porsi *Cohesion* ketika berinteraksi dengan Pengguna.

Tabel 4.7 *Use Case Scenario* Ukur Porsi *Cohesion*

Ukur Porsi *Cohesion*	
Kode Kebutuhan	SRS_F_SUMIT_003
Objective	Memberikan Pengguna layanan untuk mengukur porsi *cohesion* dari rancangan perangkat lunak yang dipilih.
Aktor	Pengguna
Pre Conditions	1. Pengguna berada pada halaman *browse file*. 2. Pengguna telah memilih sebuah berkas untuk diukur.
Main Flow	1. Pengguna menekan tombol *process*. 2. Sistem melakukan *parsing* berkas XML rancangan perangkat lunak. 3. Sistem menghitung *cohesion*. 4. Sistem menghitung nilai porsi *cohesion*. 5. Progress bar menampilkan kemajuan proses perhitungan.
Alternative Flow	1. Jika berkas yang dipilih Pengguna tidak ber*format* XML maka sistem menampilkan pemberitahuan kesalahan berkas tidak ber*format* XML. 2. Jika berkas XML tidak memiliki struktur *simple* milik Visual Paradigm maka sistem akan menampilkan pemberitahuan berkas XML tidak sesuai dengan struktur yang bisa di*parsing* SUMIT. 3. Jika tidak ada berkas yang terpilih maka sistem menampilkan pemberitahuan tidak ada berkas yang dipilih Pengguna.
Post Conditions	Sistem berhasil mengukur nilai porsi *cohesion* dan nilai *understandability*.

4. *Use case scenario* Ukur Porsi *Inheritance* (SRS_F_SUMIT_004)

   Pada Tabel 4.8 menjelaskan secara rinci perilaku dari *use case* Ukur Porsi *Inheritance* ketika berinteraksi dengan Pengguna.

Tabel 4.8 *Use Case Scenario* Ukur Porsi *Inheritance*

Ukur Porsi *Inheritance*	
Kode Kebutuhan	SRS_F_SUMIT_004
Objective	Memberikan Pengguna layanan untuk mengukur porsi *inheritance* dari rancangan perangkat lunak yang dipilih.
Aktor	Pengguna
Pre Conditions	1. Pengguna berada pada halaman *browse file*. 2. Pengguna telah memilih sebuah berkas untuk diukur.
Main Flow	1. Pengguna menekan tombol *process*. 2. Sistem melakukan *parsing* berkas XML rancangan perangkat lunak. 3. Sistem menghitung *inheritance*. 4. Sistem menghitung nilai porsi *inheritance*. 5. Progress bar menampilkan kemajuan proses perhitungan.
Alternative Flow	1. Jika berkas yang dipilih Pengguna tidak ber*format* XML maka sistem menampilkan pemberitahuan kesalahan berkas tidak ber*format* XML. 2. Jika berkas XML tidak memiliki struktur *simple* milik Visual Paradigm maka sistem akan menampilkan pemberitahuan berkas XML tidak sesuai dengan struktur yang bisa di*parsing* SUMIT. 3. Jika tidak ada berkas yang terpilih maka sistem menampilkan pemberitahuan tidak ada berkas yang dipilih Pengguna.
Post Conditions	Sistem berhasil mengukur nilai porsi *inheritance* dan nilai *understandability*.

5. *Use case scenario* Ukur *Understandability* (SRS_F_SUMIT_005)

   Pada Tabel 4.9 menjelaskan secara rinci perilaku dari *use case* Ukur Porsi *Understandability* ketika berinteraksi dengan Pengguna.

Tabel 4.9 *Use Case Scenario* Ukur *Understandability*

Ukur *Understandability*	
Kode Kebutuhan	SRS_F_SUMIT_005
Objective	Memberikan Pengguna layanan untuk mengukur nilai *understandability* dari rancangan perangkat lunak yang dipilih.
Aktor	Pengguna
Pre Conditions	1. Pengguna berada pada halaman *browse file*. 2. Pengguna telah memilih sebuah berkas untuk diukur.
Main Flow	1. Pengguna menekan tombol *process*. 2. Sistem melakukan *parsing* berkas XML rancangan perangkat lunak. 3. Sistem menghitung *Understandability*. 4. Progress bar menampilkan kemajuan proses perhitungan.
Alternative Flow	1. Jika berkas yang dipilih Pengguna tidak ber*format* XML maka sistem menampilkan pemberitahuan kesalahan berkas tidak ber*format* XML. 2. Jika berkas XML tidak memiliki struktur *simple* milik Visual Paradigm maka sistem akan menampilkan pemberitahuan berkas XML tidak sesuai dengan struktur yang bisa di*parsing* SUMIT. 3. Jika tidak ada berkas yang terpilih maka sistem menampilkan pemberitahuan tidak ada berkas yang dipilih Pengguna.
Post Conditions	Sistem berhasil mengukur nilai *understandability*.

6. *Use case scenario* Lihat Hasil (SRS_F_SUMIT_006)

   Pada Tabel 4.10 menjelaskan secara rinci perilaku dari *use case* Lihat Hasil ketika berinteraksi dengan Pengguna.

Tabel 4.10 *Use Case Scenario* Lihat Hasil

Lihat Hasil	
Kode Kebutuhan	SRS_F_SUMIT_006
Objective	Memberikan Pengguna layanan untuk melihat hasil pengukuran *understandability*, porsi *coupling*, porsi *cohesion*, dan porsi *inheritance* rancangan perangkat lunak.
Aktor	Pengguna
Pre Conditions	1. Pengguna berada pada halaman *browse file*. 2. Sistem telah melakukan proses pengukuran *understandability*, porsi *coupling*, porsi *cohesion*, dan porsi *inheritance*.
Main Flow	1. Pengguna menekan tombol navigasi ke halaman lihat hasil.
Alternative Flow	-
Post Conditions	Sistem menampilakan hasil pengukuran *understandability*, porsi *coupling*, porsi *cohesion*, porsi *inheritance*, dan perincian pengukuran *multivariate understandability metric*.

7. *Use case scenario* Lihat Ikhtisar (SRS_F_SUMIT_007)

   Pada Tabel 4.11 menjelaskan secara rinci perilaku dari *use case* Lihat Ikhtisar ketika berinteraksi dengan Pengguna.

Tabel 4.11 *Use Case Scenario* Lihat Ikhtisar

Lihat Iktisar	
Kode Kebutuhan	SRS_F_SUMIT_007
Objective	Memberikan Pengguna layanan untuk melihat hasil ikhtisar dari pengukuran *understandability* rancangan perangkat lunak.
Aktor	Pengguna
Pre Conditions	1. Pengguna berada pada halaman *browse file*. 2. Sistem telah melakukan proses pengukuran *understandability*, porsi *coupling*, porsi *cohesion*, dan porsi *inheritance*.

Main Flow	1. Pengguna menekan tombol navigasi ke halaman lihat ikhtisar.
Alternative Flow	-
Post Conditions	Sistem menampilakan hasil ikhtisar berupa hasil pengukuran *understandability*, porsi *coupling*, porsi *cohesion*, porsi *inheritance*, nilai metrik NAssoc, metrik NA, metrik NOC, total jumlah kelas, atribut, operasi, asosiasi, dan generalisasi.

8. *Use case scenario* Banding *Design* (SRS_F_SUMIT_008)

   Pada Tabel 4.12 menjelaskan secara rinci perilaku dari *use case* Banding *Design* ketika berinteraksi dengan Pengguna.

Tabel 4.12 *Use Case Scenario* Banding *Design*

Banding *Design*	
Kode Kebutuhan	SRS_F_SUMIT_008
Objective	Memberikan Pengguna layanan untuk membandingkan tingkat *understandability* antara dua buah rancangan perangkat lunak.
Aktor	Pengguna
Pre Conditions	1. Pengguna berada pada halaman banding *Design*. 2. Sistem telah memiliki salah satu hasil pengukuran *understandability* sebagai pembanding.
Main Flow	1. Sistem menampilkan nilai *understandability* pembanding. 2. Pengguna menekan tombol *browse file*. 3. Pengguna memilih sebuah berkas. 4. Pengguna menekan tombol *compare* 4. Progress bar menampilkan kemajuan proses perbandingan.
Alternative Flow	1. Jika sistem belum memiliki tingkat *understandability* pembanding maka pengguna tidak dapat melakukan *browse file* yang dibandingkan. 2. Jika berkas yang dipilih Pengguna tidak ber*format* XML maka sistem menampilkan pemberitahuan kesalahan berkas tidak ber*format* XML. 3. Jika berkas XML tidak memiliki struktur *simple* milik Visual Paradigm maka sistem akan

	menampilkan pemberitahuan berkas XML tidak sesuai dengan struktur yang bisa di*parsing* SUMIT. 4. Jika tidak ada berkas yang terpilih maka sistem menampilkan pemberitahuan tidak ada berkas yang dipilih Pengguna.
Post Conditions	Sistem menampilkan perbandingan tingkat *understandability* antara dua buah rancangan perangkat lunak.

BAB 5 PERANCANGAN SISTEM DAN IMPLEMENTASI

Perancangan yang dibahas antara lain mencakup perancangan arsitektur yang terdiri dari sequence diagram dan class diagram, perancangan algoritme dan deskripsi API/*library* yang digunakan. Kemudian perancangan antarmuka pengguna pada SUMIT berupa *mock-up* sistem. Sedangkan untuk implementasi akan dibahas antara lain mencakup spesifikasi lingkungan pengembangan dan hasil akhir penerapan sistem SUMIT.

5.1. Perancangan Arsitektur

5.1.1. Sequence Diagram

1. *Browse* Berkas (SRS_F_SUMIT_001)

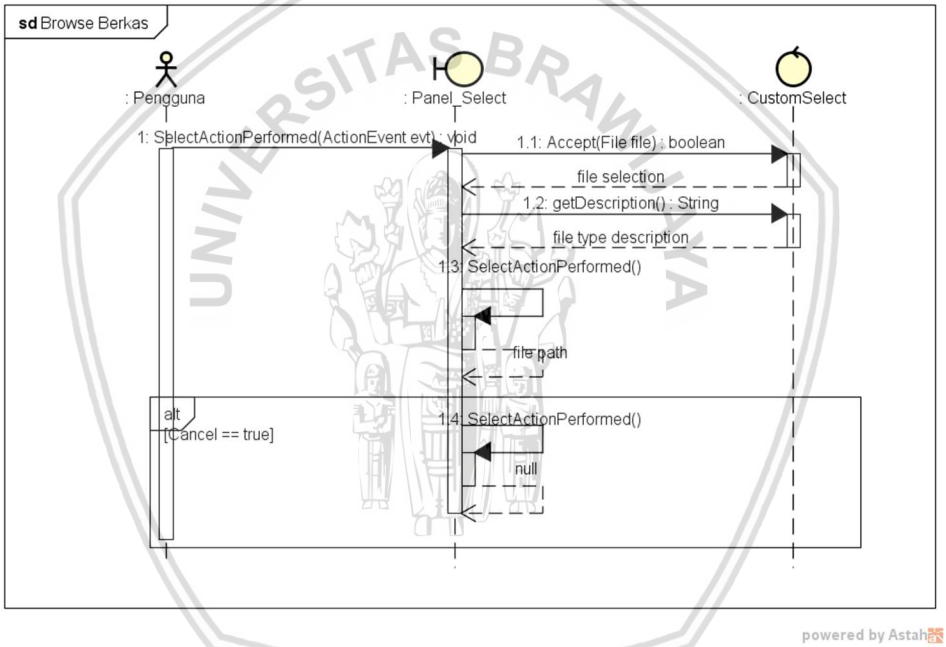


Diagram 5.1 *Sequence Diagram Browse* Berkas

Diagram 5.1 menjelaskan alur interaksi ketika aktor yaitu Pengguna melakukan *browse* berkas untuk memilih berkas rancangan perangkat lunak yang nantinya akan diukur nilai *understandabilitynya*. Pada diagram 5.1 ini terdapat dua buah objek yang terlibat yaitu Panel_Select yang bertindak sebagai *boundry* dan *CustomSelect* sebagai *controller*. Dimana tugas *Panel_Select* menjadi jembatan kepada pengguna ketika berinteraksi dengan sistem dalam memilih berkas dengan memanggil *SelectActionPerformed*. Ketika *SelectActionPerformed* dipanggil maka sistem akan menampilkan sebuah *dialog file chooser* untuk memilih berkas. Berkas yang ditampilkan secara *default* hanyalah yang bertipe XML dikarenakan bentuk berkas yang akan diolah adalah berkas bertipe XML hal ini diatur oleh *CustomSelect*. Ketika Pengguna telah memilih berkas maka Panel_Select

mendapatkan *file path* dari berkas tersebut jika tidak maka akan dikembalikan dengan nilai *null*.

2. Ukur Porsi *Coupling* (SRS_F_SUMIT_002)

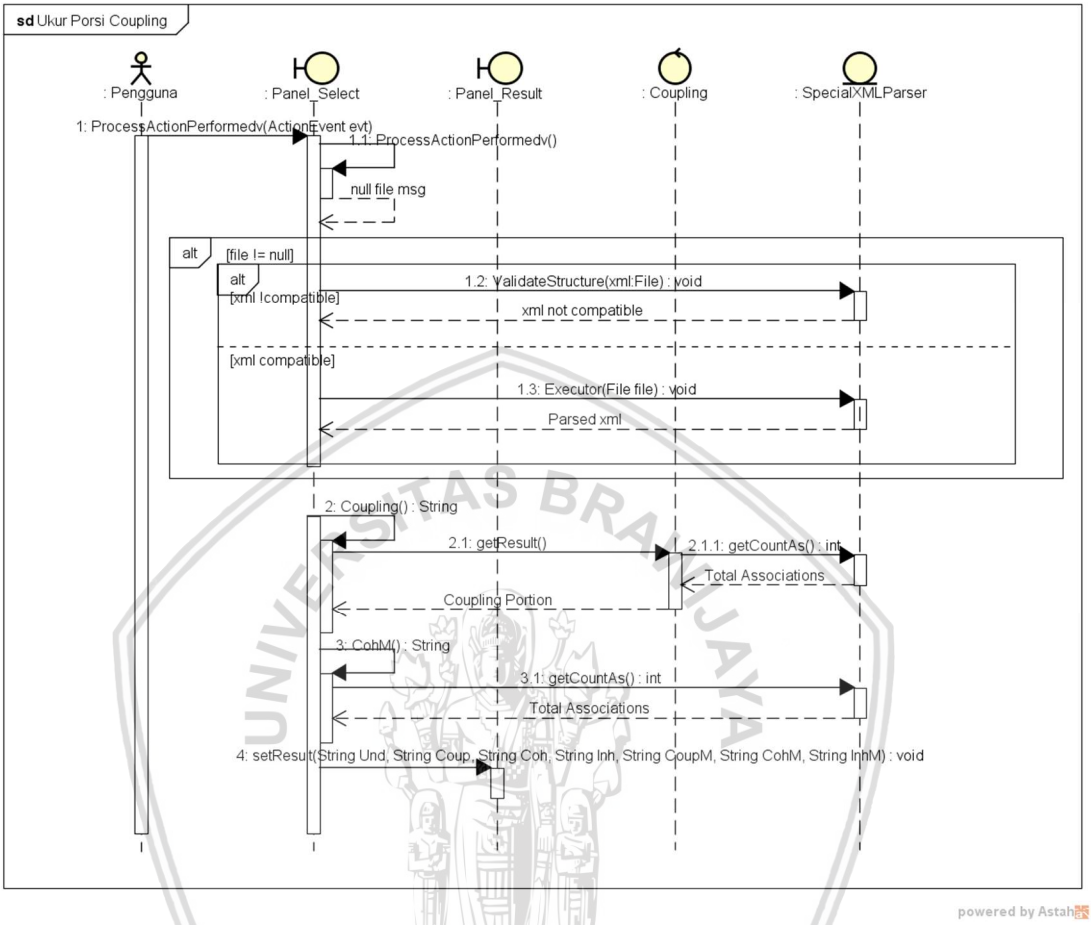


Diagram 5.2 *Sequence Diagram* Ukur Porsi *Coupling*

Diagram 5.2 menjelaskan alur interaksi ketika aktor yaitu Pengguna miminta SUMIT untuk melakukan pemgolahan berkas perangcangan perangkat lunak dalam bentuk *XML* untuk mengetahui nilai *understandability*nya dimana SUMIT akan menghitung nilai porsi *coupling* terlebih dahulu sesuai persamaan *multivariate understandability*. Pada diagram 5.2 ini terdapat empat buah objek yang terlibat yaitu *Panel_Select*, *Panel_Result*, *Coupling*, dan *SpecialXMLParser*. Pada saat pengguna memanggil *ProcessActionPerformed* dari *Panel_Select* maka berkas rancangan perangkat lunak yang dipilih akan di*parsing* dan diambil elemen yang dibutuhkan untuk menghitung nilai porsi *coupling* yang dilakukan oleh *SpecialXMLParser*. Setelah elemen yang dibutuhkan telah didapatkan kemudian *Coupling* akan mengukur porsi *coupling* dengan dipanggilanya *getResult*. Dari *getResult* maka diketahui berapa nilai porsi *coupling* yang kemudian hasilnya akan ditampilkan oleh *Panel_Result* dengan memanggil *setResult*. Proses pengukuran tidak akan dilakukan jika struktur XML tidak susai, ini divalidasi oleh objek SpecialXMLParser dengan ValidateStructure. Selain itu ketika tidak ada berkas

maka tidak akan ada proses perhitungan juga, ini dicekn langsung oleh objek Panel_Select ketika melakikan *ProcessActionPerformed*.

3. Ukur Porsi *Cohesion* (SRS_F_SUMIT_003)

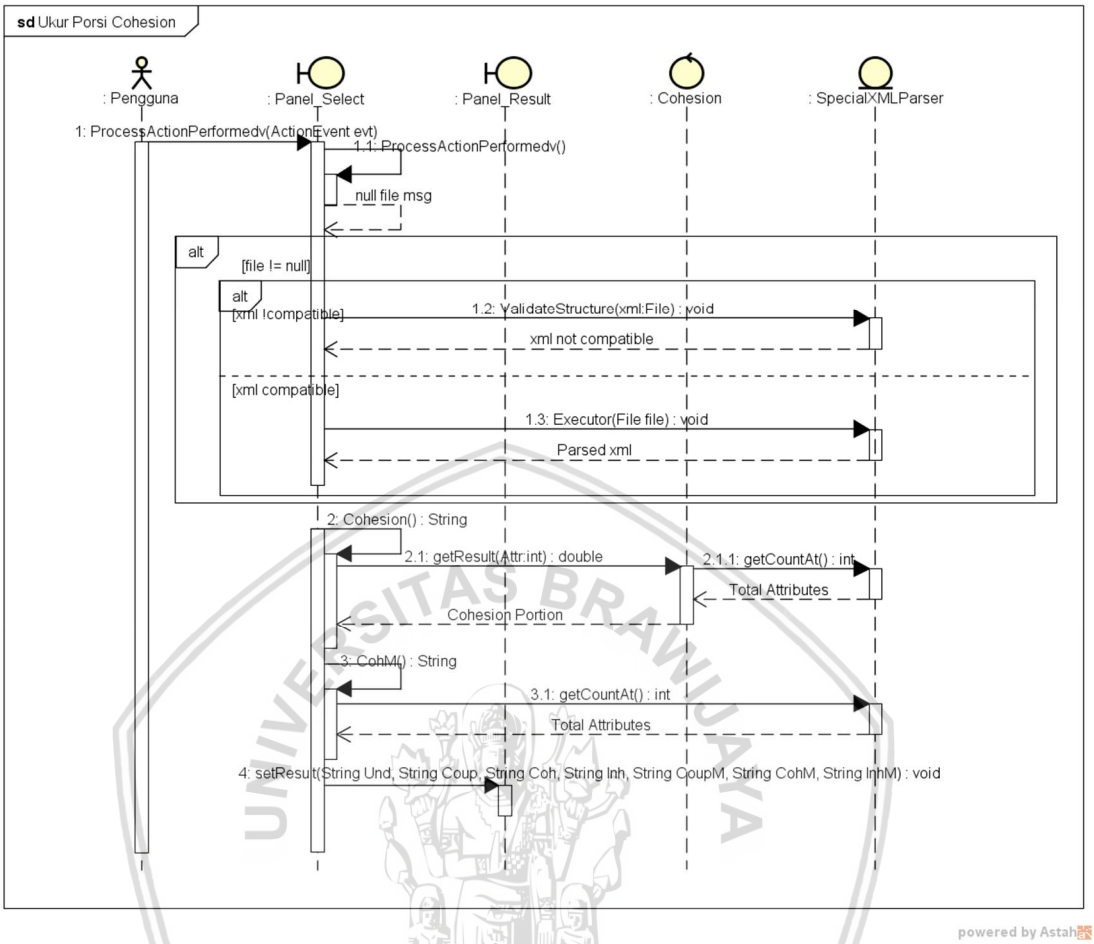


Diagram 5.3 *Sequence Diagram* UkurPorsi *Cohesion*

Diagram 5.3 menjelaskan alur interaksi ketika aktor yaitu Pengguna miminta SUMIT untuk melakukan pemgolahan berkas perangcangan perangkat lunak dalam bentuk *XML* untuk mengetahui nilai *understandability*nya dimana SUMIT akan menghitung nilai porsi *cohesion* terlebih dahulu sesuai persamaan *multivariate understandability*. Pada diagram 5.3 ini terdapat empat buah objek yang terlibat yaitu *Panel_Select*, *Panel_Result*, *Cohesion*, dan *SpecialXMLParser*. Pada saat pengguna memanggil *ProcessActionPerformed* dari *Panel_Select* maka berkas rancangan perangkat lunak yang dipilih akan di*parsing* dan diambil elemen yang dibutuhkan untuk menghitung nilai porsi *cohesion* yang dilakukan oleh *SpecialXMLParser*. Setelah elemen yang dibutuhkan telah didapatkan kemudian *Cohesion* akan mengukur porsi *cohesion* dengan dipanggilanya *getResult*. Dari *getResult* maka diketahui berapa nilai porsi *cohesion* yang kemudian hasilnya akan ditampilkan oleh *Panel_Result* dengan memanggil *setResult*. Proses pengukuran tidak akan dilakukan jika struktur XML tidak susai, ini divalidasi oleh objek SpecialXMLParser dengan ValidateStructure. Selain itu ketika tidak ada berkas maka tidak akan ada proses perhitungan juga, ini dicekn langsung oleh objek Panel_Select ketika melakikan *ProcessActionPerformed*.

4. Ukur Porsi *Inheritance* (SRS_F_SUMIT_004)

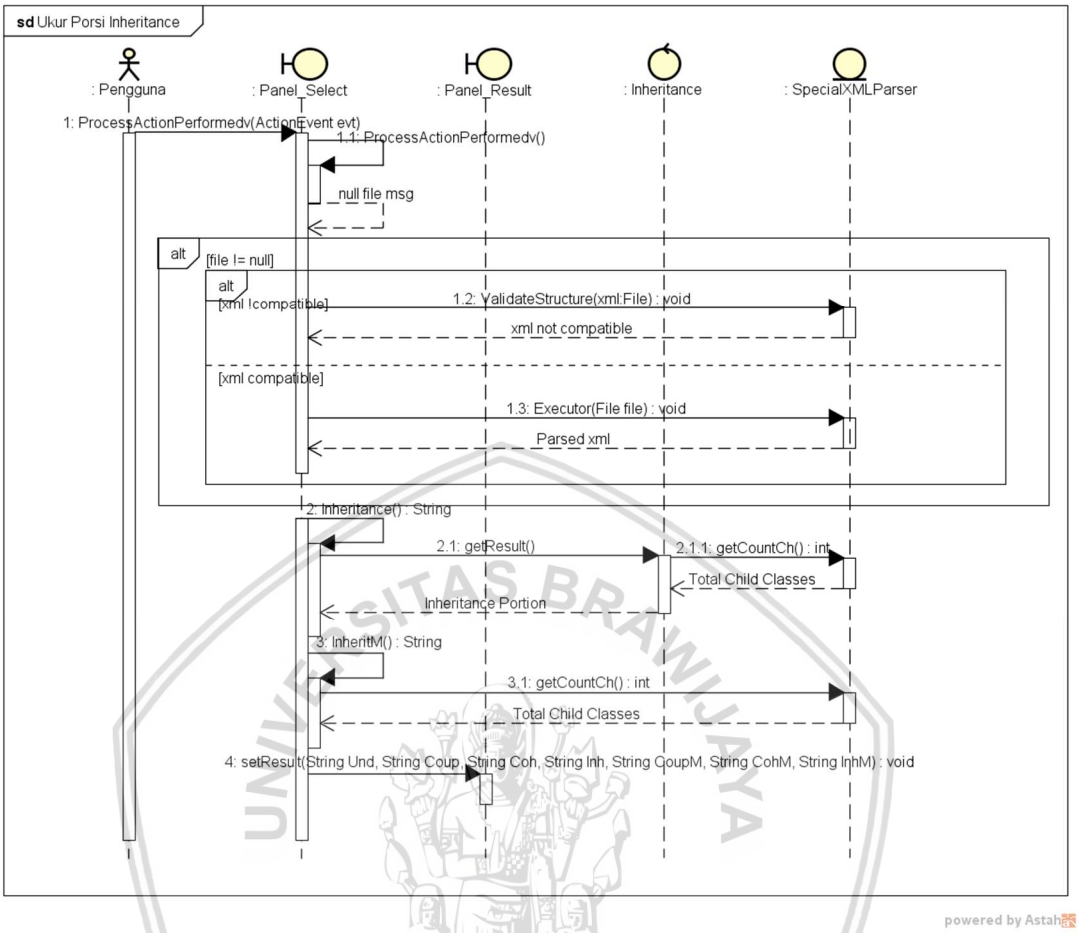


Diagram 5.4 *Sequence Diagram* Ukur Porsi *Inheritance*

Diagram 5.4 menjelaskan alur interaksi ketika aktor yaitu Pengguna miminta SUMIT untuk melakukan pemgolahan berkas perangcangan perangkat lunak dalam bentuk *XML* untuk mengetahui nilai *understandability*nya dimana SUMIT akan menghitung nilai porsi *cohesion* terlebih dahulu sesuai persamaan *multivariate understandability*. Pada diagram 5.3 ini terdapat empat buah objek yang terlibat yaitu *Panel_Select*, *Panel_Result*, *Inheritance*, dan *SpecialXMLParser*. Pada saat pengguna memanggil *ProcessActionPerformed* dari *Panel_Select* maka berkas rancangan perangkat lunak yang dipilih akan di*parsing* dan diambil elemen yang dibutuhkan untuk menghitung nilai porsi *inheritance* yang dilakukan oleh *SpecialXMLParser*. Setelah elemen yang dibutuhkan telah didapatkan kemudian *Inheritance* akan mengukur porsi *cohesion* dengan dipanggilanya *getResult*. Dari *getResult* maka diketahui berapa nilai porsi *Inheritance* yang kemudian hasilnya akan ditampilkan oleh *Panel_Result* dengan memanggil *setResult*. Proses pengukuran tidak akan dilakukan jika struktur XML tidak susai, ini divalidasi oleh objek SpecialXMLParser dengan ValidateStructure. Selain itu ketika tidak ada berkas maka tidak akan ada proses perhitungan juga, ini dicekn langsung oleh objek Panel_Select ketika melakikan *ProcessActionPerformed*.

5. Ukur *Understandability* (SRS_F_SUMITI_005)

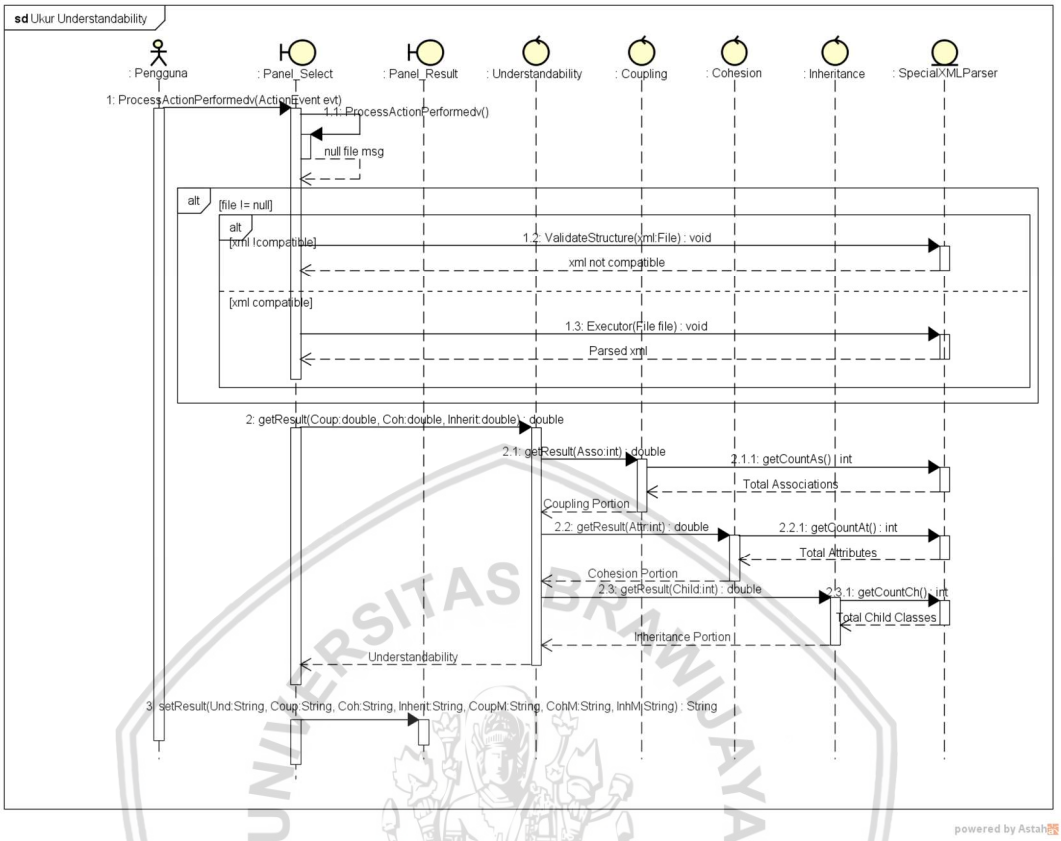


Diagram 5.5 *Sequence Diagram* Ukur *Understandability*

Diagram 5.5 menjelaskan alur interaksi ketika aktor yaitu pengguna miminta SUMIT untuk melakukan pengolahan berkas rancangan perangkat lunak dalam bentuk *XML* untuk mengetahui nilai *understandability*nya dimana SUMIT akan terlihat setelah mendapatkan nilai porsi *coupling,* nilai porsi *cohesion,* dan nilai porsi *inheritance* sesuai persamaan *multivariate understandability*. Pada diagram 5.5 ini terdapat tujuh buah objek yang terlibat yaitu *Panel_Select*, *Panel_Result*, *Coupling, Cohesion*, *Inheritance*, SpecialXMLParser dan *Understandability*. Pada saat pengguna memanggil *ProcessActionPerformed* dari *Panel_Select* dan berkas rancangan perangkat lunak telah di*parsing* oleh SpecilXMLParser dengan menggunak Executor maka *Coupling* akan memanggil *getResult* untuk mendapatkan nilai porsi *coupling*. Sementara *Cohesion* akan memanggil *getResult* untuk mendapatkan nilai porsi *cohesion* dan *Inheritance* akan memanggil *getResult* untuk mendapatkan nilai porsi *inheritance*. Setelah semua nilai porsi yang mempengaruhi *understandability* didapat selanjutmya *Undestandability* akan mengelolah data tersebut dengan memanggil *getResult* yang kemudian hasilnya akan ditampilakn oleh *Panel_Result* dengan memanggil *setResult*. Proses pengukuran tidak akan dilakukan jika struktur XML tidak susai, ini divalidasi oleh objek SpecialXMLParser dengan ValidateStructure. Selain itu ketika tidak ada berkas maka tidak akan ada proses perhitungan juga, ini dicekn langsung oleh objek Panel_Select ketika melakikan *ProcessActionPerformed*.

6. Lihat Hasil(SRS_F_SUMIT_006)

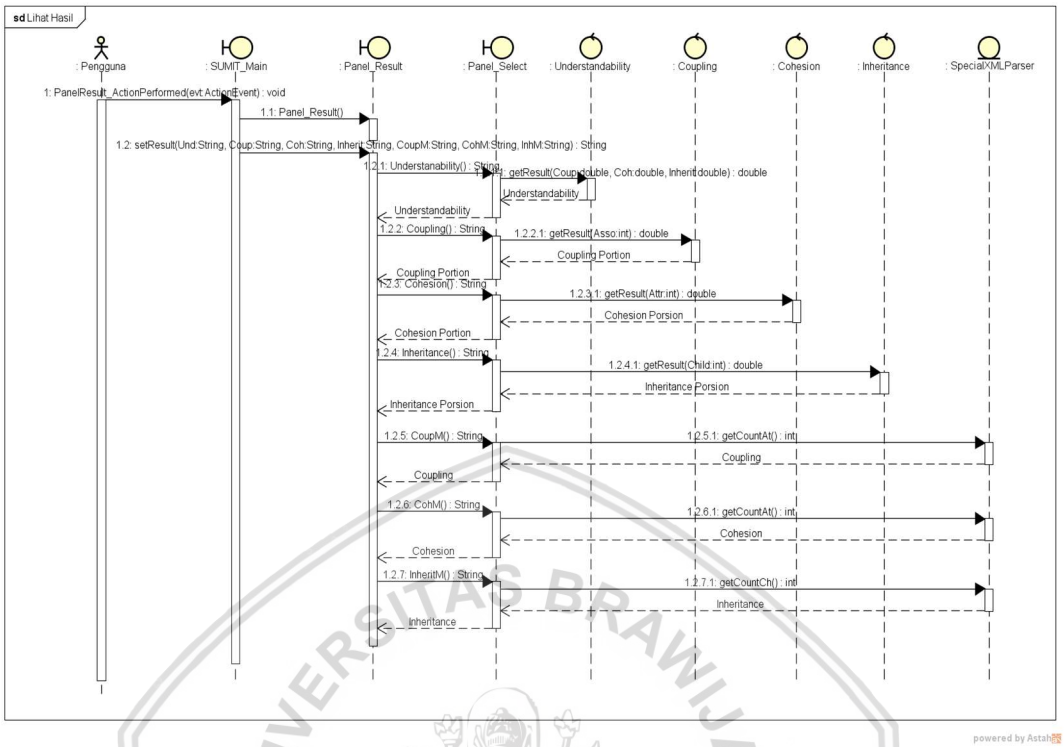


Diagram 5.6 *Sequence Diagram* Lihat Hasil

Diagram 5.6 menjelaskan alur interaksi ketika aktor yaitu pengguna miminta SUMIT untuk memperlihatkan hasil perhitungan *understandability* berkas perangcangan perangkat lunak yang di *input*. Pada diagram 5.6 ini terdapat delapan buah objek yang terlibat yaitu *Panel_Result, SUMIT_Main, Panel_Select, Understandability, Coupling, Cohesion, Inhertitance, dan SpecialXMLParser*. Ketika Panel_Result melakukan setResult maka objek *Understandability, Coupling, Cohesion, Inhertitance* melakukan getResult kemudian hasilnya akan dikembalikan ke Panel_Result untuk ditampilkan.

7. Lihat Ikhtisar (SRS_F_SUMIT_007)

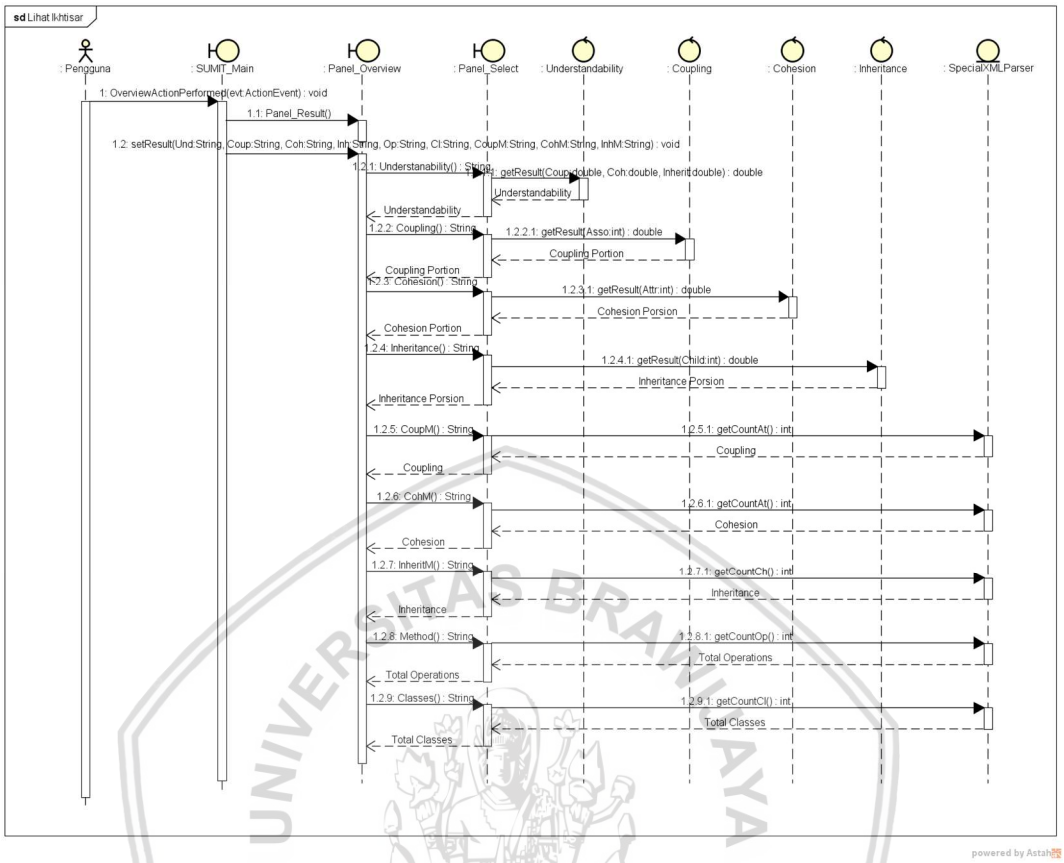


Diagram 5.7 *Sequence Diagram* Lihat Ikhtisar

Diagram 5.7 menjelaskan alur interaksi ketika aktor yaitu pengguna miminta SUMIT untuk memperlihatkan hasil ikhtisar perhitungan *understandability* berkas perangcangan perangkat lunak yang di *input*. Pada diagram 5.7 ini terdapat delapan buah objek yang terlibat yaitu *Panel_Overview, SUMIT_Main, Panel_Select, Understandability, Coupling, Cohesion, Inhertitance, dan SpecialXMLParser*. Ketika *Panel_Result* melakukan *setResul*t maka objek *Understandability, Coupling, Cohesion, Inhertitance* melakukan getResult kemudian hasilnya akan dikembalikan ke *Panel_Overview* untuk ditampilkan.

8. Banding *Design* (SRS_F_SUMIT_008)

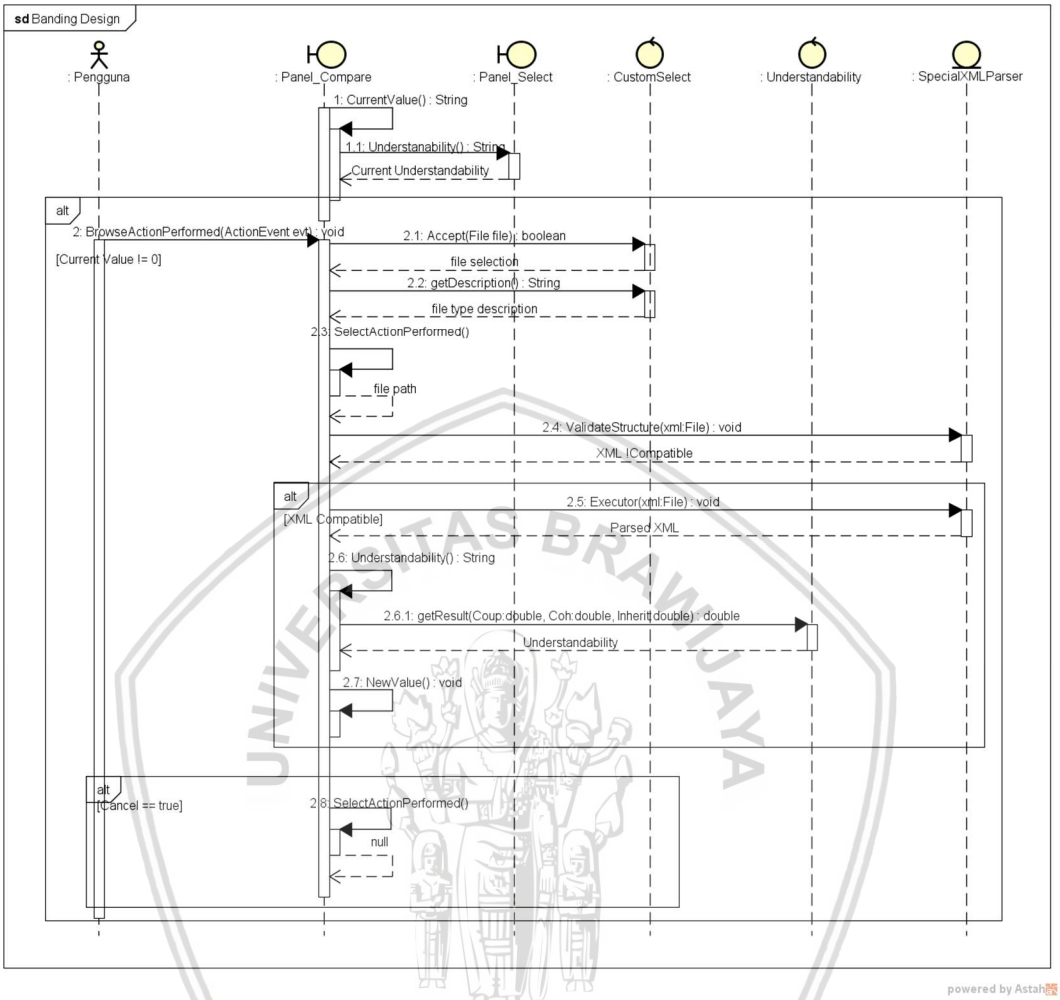


Diagram 5.8 *Sequence Diagram* Banding *Design*

Diagram 5.8 menjelaskan alur interaksi ketika aktor yaitu pengguna miminta SUMIT untuk membandingkan dua hasil perhitungan *understandability* berkas perangcangan perangkat lunak yang di input. Pada diagram 5.8 ini terdapat lima buah objek yang terlibat yaitu *Panel_Compare*, *Panel_Select*, CustomSelect, Understandability, dan *SpecialXMLParser*. Ketika *Panel_Compare* akan melakukan *BrowseActionPerformed* Panel_Compare akan melakukan CurrentValue berdasarkan *Understandability* pada *Panel_Select* jika tidak ada maka BrowseActionPerformed tidak dapat dilakukan. Jika *BrowseActionPerformed* berhasil dilakukan maka akan muncul dialog *file chooser* yang sifatnya sudah diatur oleh CustomSelect. Proses membandingkan nilai *understandability* akan dilaksanankan jika struktur XML yang terbanding benar ketika SpecialXMLParser melakukan ValidateStructure dan jika benar maka SpecialXMLParser akan melakukan Executor untuk melakukan *parsing* berkas. Ketika parsing berhasil maka Panel_Compare akan melakukan *NewValue* untuk menampilkan hasil *understandability* terbanding.

5.1.2. Class Diagram

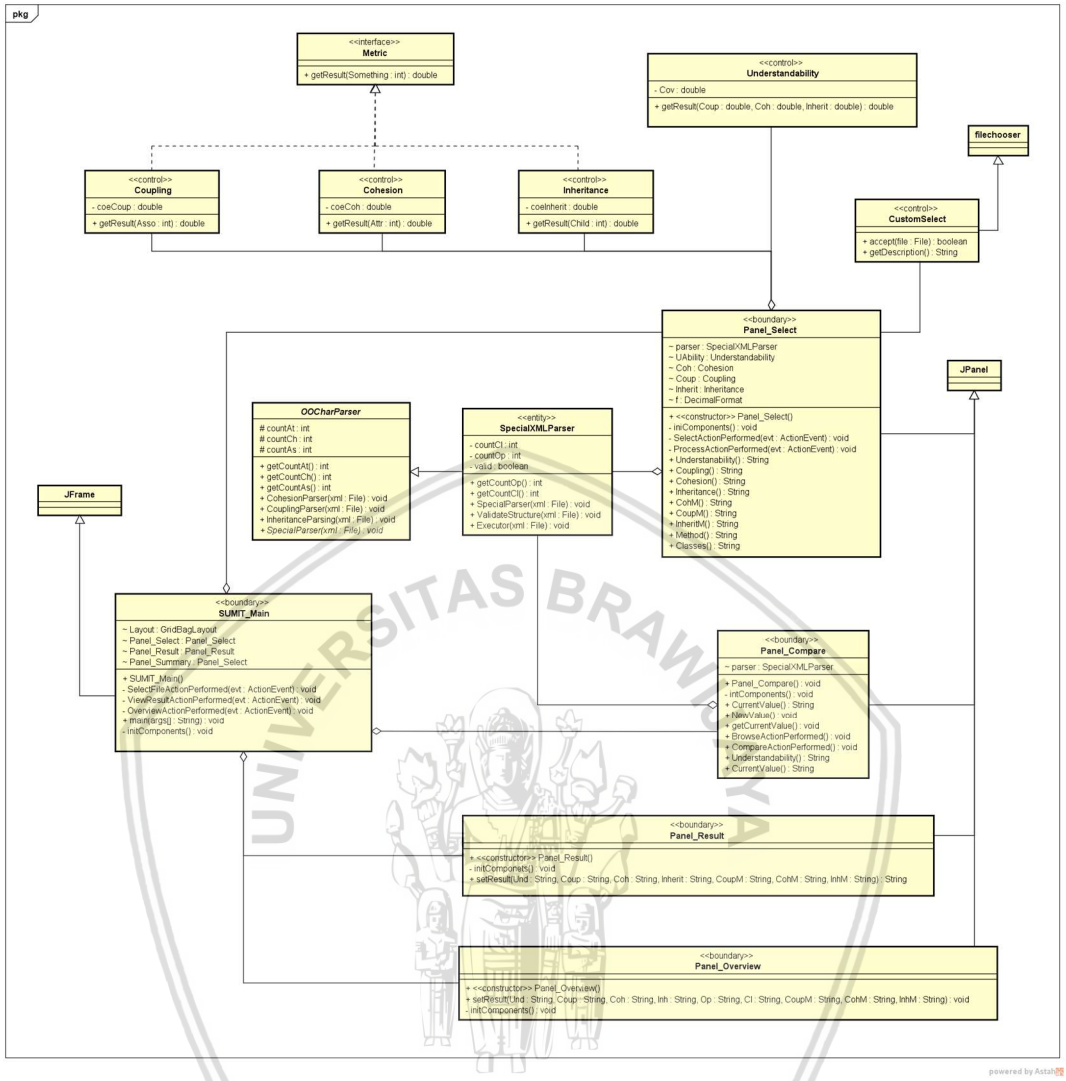


Diagram 5.9 *Class Diagram* SUMIT

Diagram 5.9 menunjukan *class diagram* pada sistem SUMIT. Dimana Diagram 5.9 ini terdiri dari sebuah kelas *interface*, sebuah kelas abstract, lima buah kelas *boundry*, enam buah kelas biasa, dan tiga buah kelas dari penggunaan API Swing untuk pembuatan GUI. Untuk lebih rincinya adalah sebagai berikut:

1. Kelas *interface Metric* yaitu kelas yang menjadi bentuk abstraksi dari kelas *Coupling*, kelas *Cohesion*, dan kelas *Inheritance*. Memiliki sebuah operasi abstrak yaitu *getResult*.
2. Kelas *control Coupling* yaitu kelas yang bertugas untuk menghitung nilai porsi *coupling* dari rancangan perangkat lunak inputan dari pengguna. Kelas ini merupakan realisasi dari kelas *interface Metric*. Kelas ini memiliki sebuah atribut *final coeCoup* yang menyimpan nilai koefisien *coupling* terhadap *understandability*. Kelas ini mempunyai sebuah operasi yang merupakan *overriding* dari kelas *interface Metric* yaitu *getResult*. Kelas ini beragregasi dengan kelas *boundry Panel_Select.*

3. Kelas *control Cohesion* yaitu kelas yang bertugas untuk menghitung nilai porsi *cohesion* dari rancangan perangkat lunak inputan dari pengguna. Kelas ini merupakan realisasi dari kelas *interface Metric*. Kelas ini memiliki sebuah atribut *final coeCoh* yang menyimpan nilai koefisien *cohesion* terhadap *understandability*. Kelas ini mempunyai sebuah operasi yang merupakan *overriding* dari kelas *interface Metric* yaitu *getResult*. Kelas ini beragregasi dengan kelas *boundry Panel_Select.*
4. Kelas *control Inheritance* yaitu kelas yang bertugas untuk menghitung nilai porsi *inheritance* dari rancangan perangkat lunak inputan dari pengguna. Kelas ini merupakan realisasi dari kelas *interface Metric*. Kelas ini memiliki sebuah atribut *final coeInherit* yang menyimpan nilai koefisien *inhetirtance* terhadap *understandability*. Kelas ini mempunyai sebuah operasi yang merupakan *overriding* dari kelas *interface Metric* yaitu *getResult*. Kelas ini beragregasi dengan kelas *boundry Panel_Select.*
5. Kelas *control CustomSelect* yaitu kelas yang memberikan sifat kepada *dialog box file chooser* untuk secara *default* menampilkan hanya berkas berbentuk *XML* saja. Kelas ini memiliki dua buah operasi yaitu *accept* dan *getDescription*. Kelas ini memiliki hubungan asosiasi dengan kelas *boundry Panel_Select*. Kelas ini menggunakan komponen *filechooser* dari API Swing.
6. Kelas *control Understandability* yaitu kelas yang berfungsi untuk menghitung nilai *understandability* dari berkas yang diinputkan Pengguna sesuai persamaan *multivariate* menghitung *understandability*. Kelas ini memiliki sebuah atribut yaitu *Cov* yang bertugas menyimpan nilai kovarian *understandability* dan memiliki sebuah operasi yaitu *getResult*. Kelas ini beragregasi dengan kelas *boundry Panel_Select.*
7. Kelas *OOCharParser* yaitu kelas absrak yang bertugas melakukan *parsing* berkas *XML* masukan dari Pengguna. Kelas ini dikhusukan untuk menguraikan elemen-elemen yang berhungungan dengan karakteristik *object-oriented design*. Kelas ini memiliki tiga buah atribut yaitu *CountAs*, *CountAt*, dan *CountCh*. Dalam kelas ini terdapat sebuah operasi abstrak yaitu SpecialParser dan enam buah operasi konkrit (biasa) yaitu *getCountAs*, *getCountAt*, *getCountCh*, *CouplingParser*, *CohesionParser*, dan *InheritanceParser*. Kelas ini merupakan *super class* bagi kelas *SpecialXMLParser*.
8. Kelas *entity SpecialXMLParser* yaitu kelas yang mewarisi sifat kelas abstrak *OOCharParser* dan bertugas sebagai eksekutor atau *implementator* untuk melakukan parsing berkas XML masukan dari Pengguna. Kelas ini melakukan *override* terhadap operasi abstrak dari kelas *OOCharParser*. Kelas ini memiliki dua buah atribut yaitu *countOp* dan *countCl*. Di dalam kelas ini terdapat lima buah operasi yaitu *getCountOp*, *getCountCl*, *Executor*, *ValidateStructure*, dan *SpecialParser*. Kelas ini beragregasi dengan kelas *boundry Panel_Select* dan *Panel_Compare*.

9. Kelas *boundry Panel_Compare* yaitu kelas yang bertugas memuat dan menampilkan antarmuka pengguna pada saat Pengguna melakukan kegiatan membandiangkan rancangan perangkat lunak. Kelas ini menggunakan komponen *panel* dari API Swing. Dalam kelas ini terdapat sebuah konstruktor dan tujuh buah operasi yaitu *Understandability, iniComponents, CurrentValuel, NewValue, getCurrentValue, BrowseActionPerformed,* dan *CompareActionPerformed*. Kelas ini beragregasi dengan kelas *entity SpecialXMLParser* dan kelas *boundry SUMIT_Main.*
10. Kelas *boundry Panel_Select* yaitu kelas yang bertugas memuat dan menampilkan antarmuka pengguna pada saat Pengguna melakukan kegiatan seleksi rancangan perangkat lunak yang akan diukur sekaligus pada saat Pengguna melakukan pengukuran. Kelas ini menggunakan komponen *panel* dari API Swing. Dalam kelas ini terdapat sebuah konstruktor dan sebelas buah operasi yaitu *Understandability, Cohesion, Coupling, Inheritance, SelectActionPerformed, ProcessActionPerformed, CoupM, InheritM, CohM, Methods,* dan *Classes*. Kelas ini beragregasi dengan kelas *entity SpecialXMLParser* dan kelas *boundry SUMIT_Main.*
11. Kelas *boundry Panel_Result* yaitu kelas yang bertugas memuat dan menampilkan antarmuka pengguna pada saat Pengguna melakukan kegiatan melihat hasil perhitungan *understandability* rancangan perangkat lunak. Kelas ini menggunakan komponen *panel* dari API Swing. Dalam kelas ini terdapat sebuah konstruktor dan dua buah operasi yaitu *initComponents* dan *setResult*. Kelas ini beragregasi dengan kelas *boundry SUMIT_Main.*
12. Kelas *boundry Panel_Overview* yaitu kelas yang bertugas memuat dan menampilkan antarmuka pengguna pada saat Pengguna melakukan kegiatan melihat ikhtisar hasil perhitungan *understandability* rancangan perangkat lunak. Kelas ini menggunakan komponen *Panel* dari API Swing. Dalam kelas ini terdapat sebuah konstruktor dan dua buah operasi yaitu *initComponents* dan *setResult*. Kelas ini beragregasi dengan kelas *boundry SUMIT_Main.*
13. Kelas *boundry SUMIT_Main* yaitu kelas yang bertindak sebagai antarmuka utama untuk bernavigasi ke panel-panel yang lain. Kelas ini menggunakan komponen *Frame* dari API Swing. Dalam kelas ini terdapat sebuah konstruktor dan enam buah operasi yaitu, *SelectFileActionPerformed*, *ViewResultActionPerfomed*, *initComponents*, *CompareActionPerformed*, *OverviewActionPerformed*, dan *main*. Kelas ini beragregasi dengan kelas *boundry Panel_Select*, *Panel_Result*, *Panel_Compare*, dan *Panel_Overview*.

5.2. Perancangan Algoritme

Perancangan algoritma merupakan perancangan lebih detail komponen dari dekomposisi sub sistem yang dimodelkan pada Diagram 5.9. Perancangan algoritma ini akan menjelaskan atribut dan operasi yang tertera pada Diagram 5.9. Dalam perancangan sistem SUMIT penulis mengambil lima buah sampel operasi utama yaitu kelas *Coupling* berfokus pada operasi *getResult*, kelas *Panel_Select* berfokus pada operasi *SelectActionPerformed*, kelas *inheritance* berfokus pada operasi *getResult*, kelas *Understandability* berfokus pada operasi *getResult*, dan kelas *SpecialXMLParser* yang berfokus pada operasi *ValidateStructure*.

5.2.1. Kelas Coupling

Terdapat 1 buah atribut pada kelas *Coupling* yang digunakan pada operasi getResult yaitu diuraikan pada Tabel 5.1 berikut:

Tabel 5.1 Atribut Kelas *Coupling*

Nama Atribut	Tipe Data	Modifier	Nilai Inisial
coeCoep	double	private	0.129

Berikut algoritma dari operasi *getResult*:

```
getResult;
FUNCTIONNAME getResult(parameter: Asso: TYPE integer) TYPE:
double;
TYPE coeCoep is double;
TYPE Asso is int;
RETURNVALUE coeCoup * Asso;
```

5.2.2. *Kelas* Panel_Select

Berikut algoritma operasi *SelectActionPerformed*:

```
SelectActionPerformed;
FUNCTIONNAME SelectActionPerformed (parameter: evt: TYPE
ActionEvent) TYPE: void;
Pb_Process Call setValue(argument: 0 TYPE String);
TYPE returnVal is integer
returnVal = FileChooser Call showOpenDialog(argument: this TYPE
Object);
IF (returnVal == JFileChooser Call APPROVE_OPTION) THEN
          TYPE file is File;
           file = FileChooser Call getSelectedFile();
           txt_File Call setText(argument: file Call
getAbsolutePath TYPE File);
ELSE txt_File Call setText(argument: null TYPE String);
FileChooser Call setSelectedFile(argument: null TYPE String);
ENDIF
```

5.2.3. *Kelas* Inheritance

Terdapat 1 buah atribut pada kelas *Inheritance* yang digunakan pada operasi getResult yaitu diuraikan pada Tabel 5.2 berikut:

Tabel 5.2 Atribut Kelas *Inheritance*

Nama Atribut	Tipe Data	Modifier	Nilai Inisial
coeInherit	double	private	0.3405

Berikut algoritma operasi getResult:

```
getResult;
FUNCTIONNAME getResult(parameter: Gene: TYPE integer) TYPE:
double;
TYPE coeInherit is double;
TYPE Gene is int;
RETURNVALUE coeInherit * Gene;
```

5.2.4. *Kelas* Understandability

Terdapat 1 buah atribut pada kelas *Understandability* yang digunakan pada operasi getResult yaitu diuraikan pada Tabel 5.3 berikut:

Tabel 5.3 Atribut Kelas *Understandability*

Nama Atribut	Tipe Data	Modifier	Nilai Inisial
Cov	double	private	1.33515

Berikut algoritme operasi getResult:

```
getResult;
FUNCTIONNAME getResult(parameters: Coup: TYPE double, Coh: TYPE
double, Inherit: TYPE double,) TYPE: double;
TYPE Cov is double;
TYPE Coup is double;
TYPE Coh is double;
TYPE Inheritis double;
TYPE understandability is double;
IF Coup != 0 || Coh !=0 || Inherit != 0
                THEN understandability = Cov + Coup + Coh +Inherit;
ELSE understandability = 0 ;
ENDIF
RETURNVALUE understandability;
```

5.2.5. Kelas *SpecialXMLParser*

Terdapat 1 buah atribut pada kelas *SpecialXMLParser* yang digunakan pada operasi *ValidateStructure* yaitu diuraikan pada Tabel 5.4 berikut:

Tabel 5.4 Atribut Kelas *SpecialXMLParser*

Nama Atribut	Tipe Data	Modifier	Nilai Inisial
valid	boolean	private	-

Berikut algoritme operasi *ValidateStructure*:

```
ValidateStructure;
FUNCTIONNAME ValidateStructure (parameter: xml: TYPE File) TYPE:
boolean;
TYPE dbFactory is DocumentBuilderFactory;
TYPE dBuilderis DocumentBuilder;
TYPE doc is Document;
TYPE strc is NodeList;

dbFactory = DocumentBuilderFactory CALL newInstance();
dBuilder = dbFaktory CALL newDocumentBuilder();
doc = dBuilder CALL parse(argument: xml TYPE File);

doc CALL getDocumentElement() CALL normalize();
doc CALL getDocumentElement();

strc = doc CALL getElementsByTagName("Project");
TYPE nNode is Node

nNode = strc CALL item(argument: 0);

TYPE eElement is Element;

eElement = (Element) nNode;

IF    (!"simple"    =    eElement    CALL    getAttribute(argument:
Xml_structure TYPE String))

      THEN valid = false;

ELSE valid = true;

ENDIF;

return valid;
```

5.3. Deskripsi API/*Library*

Dalam pengembangan sistem SUMIT penulis menggunakan API/*library* seperti JAXP, Java AWT, Java Swing, Java DecimalFormat dan Java IO. Untuk deskripsi lebih lengkapnya dapat dilihat pada Tabel 5.5 berikut:

Tabel 5.5 Deskripsi API/*Library*

Nama API/*Library*	Deskripsi
JAXP	Tipe XML parser yang digunakan dalam bahasa pemorograman Java untuk isi dokumen XML.
Java AWT	Merupakan paket *Java* untuk membuat grafik antarmuka pengguna dua dimensi. Dalam pengembangan SUMIT, Java AWT digunakan mengelolah *layout container* dan *event-handling*.
Java Swing	Merupakan paket *Java* untuk membuat grafik antarmuka pengguna. Perbedaanya dengan *Java* AWT, Java Swing lebih mutakhir dan memiliki komponen yang lebih fleksibel. Dalam membuat antarmuka pengguna SUMIT Java Swing dipakai sebagian besar dibanding Java AWT.
Java DecimalFormat	Digunakan untuk mengatur pola *output* perhitungan metrik menjadi (#.##)
Java IO	Digunakan untuk mengatur input dan output untuk pemilihan file pada saat menggunakan *file chooser*.

5.4. Perancangan Antarmuka Pengguna

Dalam pengembangan SUMIT antarmuka pengguna dibagi kedalam lima jenis halaman yaitu halaman utama yang berjenis *frame* (window), halaman seleksi berkas dan memproses hitungan yang berjenis panel, halaman lihat hasil yang berjenis panel, halaman membandingankan rancangan yang berjenis panel dan halaman lihat ikhtisar yang berjenis panel. Berikut ini merupakan perancangan antarmuka SUMIT berupa *mock-up* yang dirancang menggunakan *Layout* pada *Draw.io*

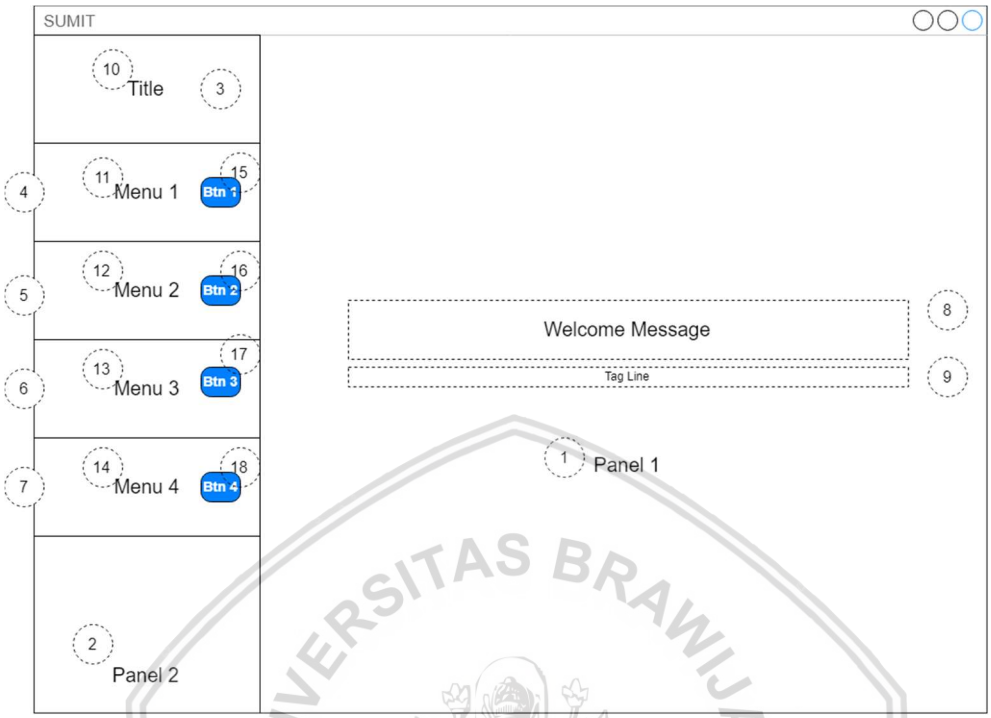
5.4.1. Antarmuka Pengguna Halaman Utama

Pada Tabel 5.6 berikut ini adalah keterangan komponen yang harus ada pada halaman utama SUMIT:

Tabel 5.6 Keterangan Antarmuka Pengguna Halaman Utama

No	Nama Komponen	Tipe Komponen	Keterangan
1	Panel Utama	Panel	Kontainer untuk menampung kompenen pada halaman utama
2	*Footer*	Panel	Kontainer footer
3	*Header*	Panel	Kontainer untuk menampung nama sistem
4	Panel Menu 1	Panel	Kontainer untuk menampung menu *Select File*
5	Panel Menu 2	Panel	Kontainer untuk menampung menu *View Result*
6	Panel Menu 3	Panel	Kontainer untuk menampung menu *Compare*
7	Panel Menu 4	Panel	Kontainer untuk menampung menu *Overview*
8	*Welcome Message*	Label	Tulisan pesan selamat datang ketika sistem pertama kali dijalankan
9	*Tag Line*	Label	Tulisan *tag line* dari sistem
10	*Title*	Label	Nama sistem
11	Menu 1	Label	Petunjuk keberadaan menu *Select File*
12	Menu 2	Label	Petunjuk keberadaan menu *View Result*
13	Menu 3	Label	Petunjuk keberadaan menu *Compare*
14	Menu 4	Label	Petunjuk keberadaan menu *Overview*
15	*Nav Button* 1	*Button*	Tombol untuk navigasi ke halaman seleksi berkas
16	*Nav Button* 2	*Button*	Tombol untuk navigasi ke halaman lihat hasil
17	*Nav Button* 3	*Button*	Tombol untuk navigasi ke halaman membandingakan rancangan
18	*Nav Button* 4	*Button*	Tombol untuk navigasi ke halaman lihat ikhtisar

Pada Gambar 5.1 berikut ini adalah *mock-up* perancangan antarmuka pengguna pada halaman utama SUMIT:



Gambar 5.1 *Mock-Up* Halaman Utama

5.4.2. Antarmuka Pengguna Halaman Seleksi Berkas

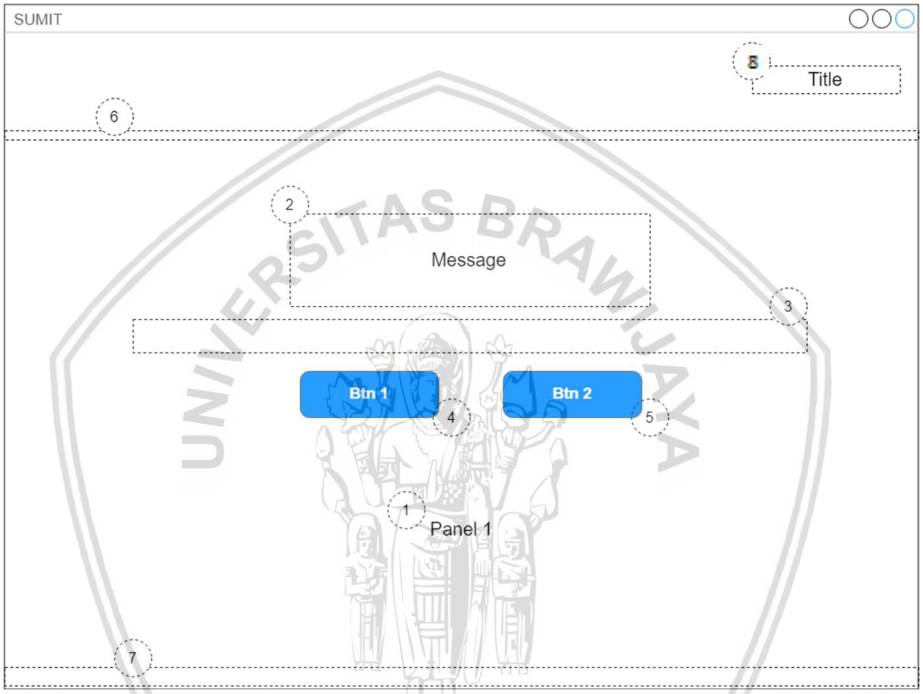
Pada Tabel 5.7 berikut ini adalah keterangan komponen yang harus ada pada halaman seleksi berkas SUMIT:

Tabel 5.7 Keterangan Antarmuka Pengguna Halaman Seleksi Berkas dan Pengukuran

No	Nama Komponen	Tipe Komponen	Keterangan
1	Panel Utama	Panel	Kontainer untuk menampung kompenen pada seleksi berkas
2	Pesan Instruksi	Label	Pesan yang memberitahukan pengguna lokasi untuk melakukan seleksi berkas
3	*File Path Field*	*Text Field*	Kotak isian untuk mengisi keberadaan berkas yang akan dipilih
4	*Browse Button*	*Button*	Tombol untuk melakukan seleksi berkas
5	*Process Button*	*Button*	Tombol untuk melakukan pengukuran berkas yang dipilih pengguna

6	Pembatas	*Separator*	Pembatas antara bagian *header* panel dan bagian *body* panel
7	*Progress Bar*	*Progress Bar*	Bilah Kemajuan yang merepresentasikan berjalannya proses perhitungan
8	*Title*	Label	Judul halaman

Pada Gambar 5.2 berikut ini adalah *mock-up* perancangan antarmuka pengguna pada halaman seleksi berkas SUMIT:



Gambar 5.2 *Mock-Up* Halaman Seleksi Berkas dan Pengukuran

5.4.3. Antarmuka Pengguna Halaman Lihat Hasil

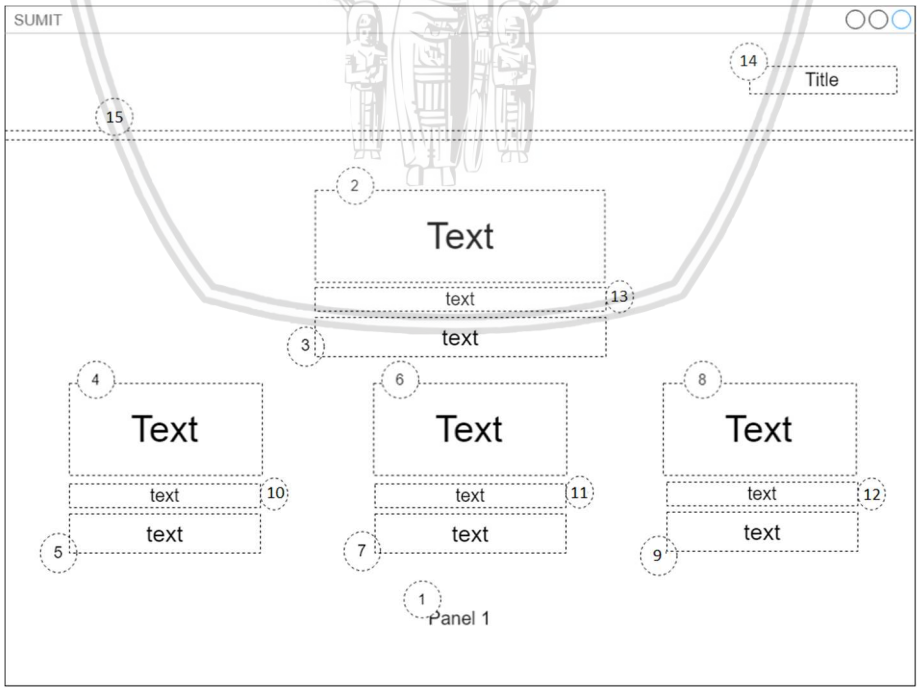
Pada Tabel 5.8 berikut ini adalah keterangan komponen yang harus ada pada halaman Lihat Hasil SUMIT:

Tabel 5.8 Keterangan Antarmuka Pengguna Halaman Lihat Hasil

No	Nama Komponen	Tipe Komponen	Keterangan
1	Panel Utama	Panel	Kontainer untuk menampung kompenen pada halaman lihat hasil
2	Hasil *understandability*	*Text Field*	Nilai *understandability*
3	Penunjuk hasil 1	Label	Penunjuk hasil *understandability*
4	Hasil *porsi cohesion*	*Text Field*	Nilai porsi *cohesion*

5	Penunjuk hasil 2	Label	Penunjuk hasil *cohesion*
6	Hasil *porsi coupling*	*Text Field*	Nilai porsi *coupling*
7	Penunjuk hasil 3	Label	Penunjuk hasil *coupling*
8	Hasil *porsi inheritance*	*Text Field*	Nilai porsi *inheritance*
9	Penunjuk hasil 4	Label	Penunjuk hasil *inheritance*
10	Perincian perhitungan 1	Label	Perincian perhitungan *cohesion*
11	Perincian perhitungan 2	Label	Perincian perhitungan *coupling*
12	Perincian perhitungan 3	Label	Perincian perhitungan *inheritance*
13	Perincian perhitungan 3	Label	Perincian perhitungan *understandability*
14	*Title*	Label	Judul halaman
15	Pembatas	*Separator*	Pembatas antara bagian *header* panel dan bagian *body* panel

Pada Gambar 5.3 berikut ini adalah *mock-up* perancangan antarmuka pengguna pada halaman lihat hasil SUMIT:



Gambar 5.3 *Mock-Up* Halaman Lihat Hasil

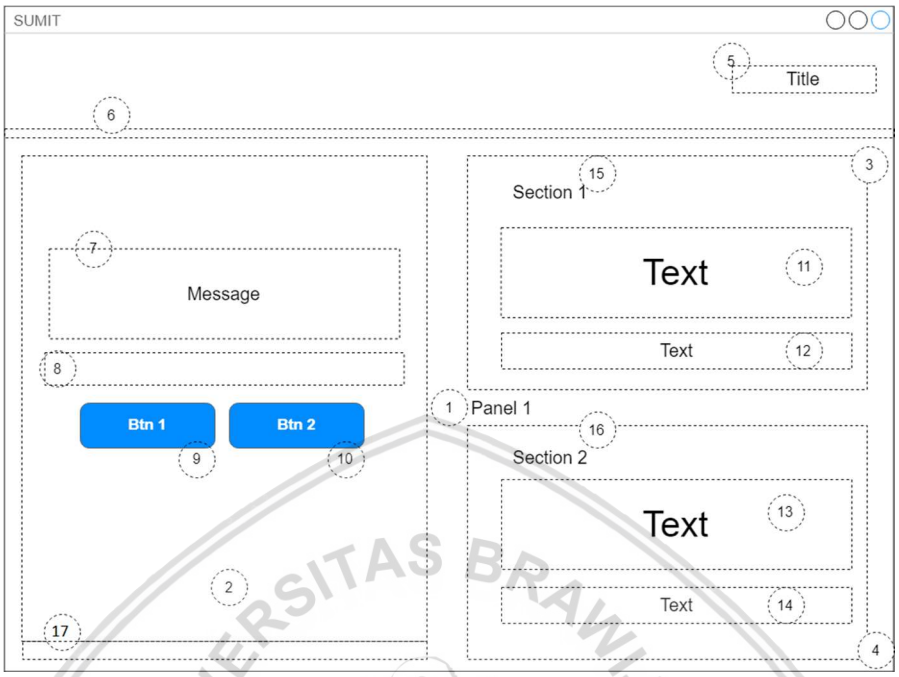
5.4.4. Antarmuka Pengguna Halaman Membandingakan Perancangan

Pada Tabel 5.9 berikut ini adalah keterangan komponen yang harus ada pada halaman Lihat Hasil SUMIT:

Tabel 5.9 Keterangan Antarmuka Pengguna Halaman Membandingkan Perancangan

No	Nama Komponen	Tipe Komponen	Keterangan
1	Panel Utama	Panel	Kontainer untuk menampung kompenen pada halaman membandingkan rancangan
2	Panel Seleksi	Panel	Nilai *understandability*
3	Panel Pembanding	Panel	Penunjuk hasil *understandability*
4	Panel Terbanding	Panel	Nilai porsi *cohesion*
5	*Title*	Label	Judul halaman
6	Pembatas	*Separator*	Pembatas antara bagian *header* panel dan bagian *body* panel
7	Pesan Instruksi	Label	Pesan yang memberitahukan pengguna lokasi untuk melakukan seleksi berkas
8	*File Path Field*	*Text Field*	Kotak isian untuk mengisi keberadaan berkas yang akan dipilih
9	*Browse Button*	*Button*	Tombol untuk melakukan seleksi berkas
10	*Compare Button*	*Button*	Tombol untuk melakukan pembandingan berkas tingkat *understandability*
11	Hasil Pembanding	Label	Tingkat *understandability* pembanding
12	Nama Hasil 1	Label	Penunjuk *understandability*
13	Hasil Terbanding	Label	Tingkat *understandability* terbanding
14	Nama Hasil 2	Label	Penunjuk *understandability*
15	Judul *Section* 1	Label	Judul *section* pembanding
16	Judul *Section* 2	Label	Judul *section* terbanding
17	*Progress Bar*	*Progress Bar*	Bilah Kemajuan yang merepresentasikan berjalannya proses perhitungan

Pada Gambar 5.4 berikut ini adalah *mock-up* perancangan antarmuka pengguna pada halaman membandingakn rancangan SUMIT:



Gambar 5.4 *Mock-Up* Halaman Membandingkan Rancangan

5.4.5. Antarmuka Pengguna Halaman Lihat Ikhtisar

Pada Tabel 5.10 berikut ini adalah keterangan komponen yang harus ada pada halaman Lihat Hasil SUMIT:

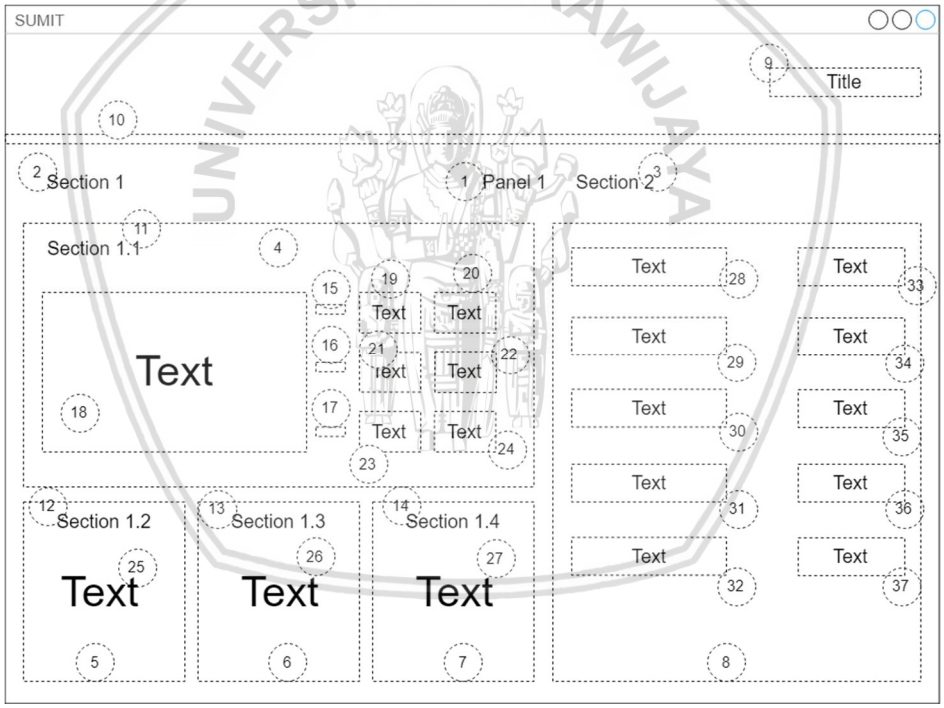
Tabel 5.10 Keterangan Antarmuka Pengguna Halaman Lihat Ikhtisar

No	Nama Komponen	Tipe Komponen	Keterangan
1	Panel Utama	Panel	Kontainer untuk menampung kompenen pada halaman lihat ikhtisar
2	Nama *Section* 1	Label	Nama penunjuk *section* yang berkaitan dengan nilai hasil perhitungan menggunakan metrik
3	Nama *Section* 2	Label	Nama penunjuk *section* yang berkaitan dengan nilai karakteristik OO pada rancangan terkait *understandability*
4	Panel Metrik *Understandability*	Panel	Kontainer untuk menampung nilai terkait *multivariate understandability metric*
5	Panel Metrik NAssoc	Panel	Kontainer untuk menampung nilai terkait metrik NAssoc
6	Panel Metrik NA	Panel	Kontainer untuk menampung nilai terkait metrik NA

7	Panel Metrik NOC	Panel	Kontainer untuk menampung nilai terkait metrik NOC
8	Panel Info Rancangan	Panel	Kontainer untuk menampung nilai terkait info rancangan
9	*Title*	Label	Judul halaman
10	Pembatas	*Separator*	Pembatas antara bagian *header* panel dan bagian *body* panel
11	Nama *Section* 1.1	Label	Nama penunjuk letak nilai *understandability*
12	Nama *Section* 1.2	Label	Nama penunjuk letak nilai metrik NAssoc
13	Nama *Section* 1.3	Label	Nama penunjuk letak nilai metrik NA
14	Nama *Section* 1.4	Label	Nama penunjuk letak nilai metrik NOC
15	Penunjuk 1	*Separator*	Garis penunjuk porsi *coupling*
16	Penunjuk 2	*Separator*	Garis penunjuk porsi *cohesion*
17	Penunjuk 3	*Separator*	Garis penunjuk porsi *inheritance*
18	Nilai *Understandability*	Label	Tampilan nilai porsi *understandability*
19	Nilai Porsi *Coupling*	Label	Tampilan nilai porsi *coupling*
20	Penunjuk Nilai Porsi *Coupling*	Label	Nama penunjuk letak porsi *coupling*
21	Nilai Porsi *Cohesion*	Label	Tampilan nilai porsi *cohesion*
22	Penunjuk Nilai Porsi *Cohesion*	Label	Nama penunjuk letak porsi *cohesion*
23	Nilai Porsi *Inheritance*	Label	Tampilan nilai porsi *inheritance*
24	Penunjuk Nilai Porsi *Inheritance*	Label	Nama penunjuk letak porsi *inheritance*
25	Nilai Metrik NAssoc	Label	Tampilan nilai metrik NAssoc
26	Nilai Metrik NA	Label	Tampilan nilai metrik NA
27	Nilai Metrik NOC	Label	Tampilan nilai metrik NOC
28	Penunjuk Nilai Total Kelas	Label	Nama penunjuk letak nilai total kelas
29	Penunjuk Nilai Total Atribut	Label	Nama penunjuk letak nilai total atribut
30	Penunjuk Nilai Total Operasi	Label	Nama penunjuk letak nilai total operasi
31	Penunjuk Nilai Total Asosiasi	Label	Nama penunjuk letak nilai total asosiasi

32	Penunjuk Nilai Total Generalisasi	Label	Nama penunjuk letak nilai total generalisasi
33	Nilai Total Kelas	Label	Tampilan nilai total jumlah kelas dari rancangan
34	Nilai Total Atribut	Label	Tampilan nilai total jumlah atribut dari rancangan
35	Nilai Total Operasi	Label	Tampilan nilai total jumlah operasi dari rancangan
36	Nilai Total Asosiasi	Label	Tampilan nilai total jumlah asosiasi dari rancangan
37	Nilai Total Generalisasi	Label	Tampilan nilai total jumlah generalisasi dari rancangan

Pada Gambar 5.5 berikut ini adalah *mock-up* perancangan antarmuka pengguna pada halaman lihat ikhtisar SUMIT:



Gambar 5.5 *Mock-Up* Halaman Lihat Ikhtisar

5.5. Implementasi

Tahapan implementasi perangkat lunak yang dilakukan terdiri dari penjelasan tentang spesifikasi sistem, lingkungan pengembangan sistem, batasan implementasi dan implementasi yang disajikan dalam bentuk *user interface*.

5.5.1. Spesifikasi Sistem

Aplikasi perangkat lunak SUMIT untuk menghitung *understandability* dikembangkan berdasarkan algoritme yang sesuai dengan *multivariate*

undestandability metric pada persamaan (1) (Nazir, Khan, & Mustafa, 2010). *Understandability* dilihat berdasarkan pengaruh dari *coupling*, *cohesion*, dan *inheritance* dari sebuah rancangan perangkat lunak. *Metric* yang telibat dalam persamaan (1) adalah NA untuk *cohesion*, NAssoc untuk *coupling*, dan NOC untuk *inheritance*. Berikut ini spesifikasi sistem berdasarkan lingkungan perangkat lunak, sistem operasi dan lingkungan perangkat keras.

5.5.1.1. Lingkungan Perangkat Lunak

Pada pengembangan aplikasi perangkat lunak SUMIT ini didukung oleh perangkat lunak pada Tabel 5.11 berikut :

Tabel 5.11 Lingkungan Perangkat Lunak

IDE	NetBeans 8.2
Bahasa Pemrograman	Java
Framework/API/*Library*	Swing, AWT, JAXP, IO
XML Generator	Visual Paradigm 14.2 Standard Edition
Modeling Tool	Astah Community 7.1.0/f2c212, Draw.io

5.5.1.2. Lingkungan Sistem Operasi

Pengembangan aplikasi perangkat lunak SUMIT ini menggunakan Sistem Operasi Windows 10 Pro versi 10.0.15063.

5.5.1.3. Lingkungan Perangkat Keras

Pada pengembangan aplikasi perangkat lunak SUMIT ini didukung oleh perangkat lunak pada Tabel 5.12 berikut :

Tabel 5.12 Lingkungan Perangkat Keras

System Model	Asus Vivobook 14 X405UQ-BV567
Processor	Intel CORE i5 7th Generation 2.50GHz
Graphic Card	NVIDIA GEFORCE 940MX
Memory	4 GB
HDD	1TB

5.5.2. Batasan Implementasi

Berikut ini merupakan uraian batasan dalam implementasi aplikasi perangkat lunak SUMIT:

1. Masukan berupa dokumen XML yang memiliki struktur *simple* milik Visual Paradigm.
2. Rancangan perangkat lunak hanyalah berbentuk *class diagram*.

3. Sistem tidak memberikan batasan nilai understandability yang aman hanya melakukan pengukuran saja.

5.5.3. Kode Sumber

Berikut ini merupakan contoh kode sumber dalam implementasi aplikasi perangkat lunak SUMIT. Contoh yang diberikan meliputi kode sumber yang berasal dari kelas *Understandability* dan kelas SpecialXMLParser.

Berikut pada Tabel 5.13 ada contoh kode sumber dari kelas *understandability*:

Tabel 5.13 Kode Sumber Kelas *Understandability*

No	Understandability.java
1	`public class Understandability {`
2	`    private final double Cov = 1.33515;`
3	`    public double getResult(double Coup, double Coh, double`
4	`Inherit) {`
5	`        double understandability = 0;`
6	`        if(Coup != 0 || Coh !=0 || Inherit != 0){`
7	`        understandability = Cov + Coup + Coh +Inherit;`
8	`        }else{`
9	`            understandability = 0;`
10	`        }`
11	`        return understandability;`
12	`    }`
13	`}`

Berikut pada Tabel 5.14 ada contoh kode sumber dari kelas *SpecialXMLParser*:

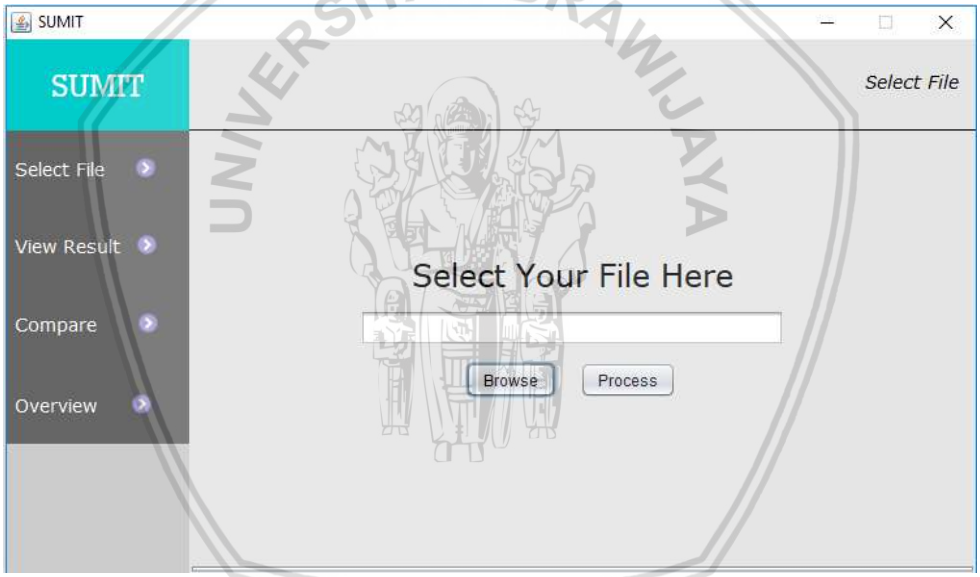
No	SpecialXMLParser.java
1	`public class SpecialXMLParser extends OOCharParser {`
2	`    private int countOp, countCl;`
3	`    private boolean valid;`
4	`    public int getCountOp() {`
5	`        return countOp;`
6	`    }`
7	`    public int getCountCl() {`
8	`        return countCl;`
9	`    }`
10	`    public void SpecialParser(File xml) throws`
11	`ParserConfigurationException, SAXException, IOException {`
12	`        countOp = 0;`
13	`        countCl = 0;`
14	`        DocumentBuilderFactory dbFactory =`
15	`DocumentBuilderFactory.newInstance();`
16	`        DocumentBuilder dBuilder = dbFactory.newDocumentBuilder();`
17	`        Document doc = dBuilder.parse(xml);`
18	`        doc.getDocumentElement().normalize();`
19	`        doc.getDocumentElement();`
20	`        NodeList opList = doc.getElementsByTagName("Operation");`
21	`        NodeList clList = doc.getElementsByTagName("Class");`

```
22          for (int temp = 0;
23                  temp < opList.getLength();
24                  temp++) {
25              Node nNode = opList.item(temp);
26              Element eElement = (Element) nNode;
27              if (eElement.getAttribute("Name") != null) {
28                  countOp++;
29              }
30          }
31
32          for (int temp = 0;
33                  temp < clList.getLength();
34                  temp++) {
35              Node nNode = clList.item(temp);
36              Element eElement = (Element) nNode;
37              if (eElement.getAttributeNode("AttributeSortType") !=
38  null) {
39                  countCl++;
40              }
41          }
42
43      }
44
45      public    boolean    ValidateStructure(File    xml)    throws
46  ParserConfigurationException, SAXException, IOException {
47          DocumentBuilderFactory              dbFactory            =
48  DocumentBuilderFactory.newInstance();
49          DocumentBuilder dBuilder = dbFactory.newDocumentBuilder();
50          Document doc = dBuilder.parse(xml);
51          doc.getDocumentElement().normalize();
52          doc.getDocumentElement();
53          NodeList strc = doc.getElementsByTagName("Project");
54          Node nNode = strc.item(0);
55          Element eElement = (Element) nNode;
56          if
57  (!"simple".equalsIgnoreCase(eElement.getAttribute("Xml_structure"
58  ))) {
59              valid = false;
60          } else {
61              valid = true;
62          }
63          return valid;
64      }
65
66      public       void       Executor(File       xml)       throws
67  ParserConfigurationException, SAXException, IOException {
68          CohesionParser(xml);
69          CouplingParser(xml);
70          InheritanceParser(xml);
71          SpecialParser(xml);
72      }
73  }
```

5.5.4. Hasil Implementasi

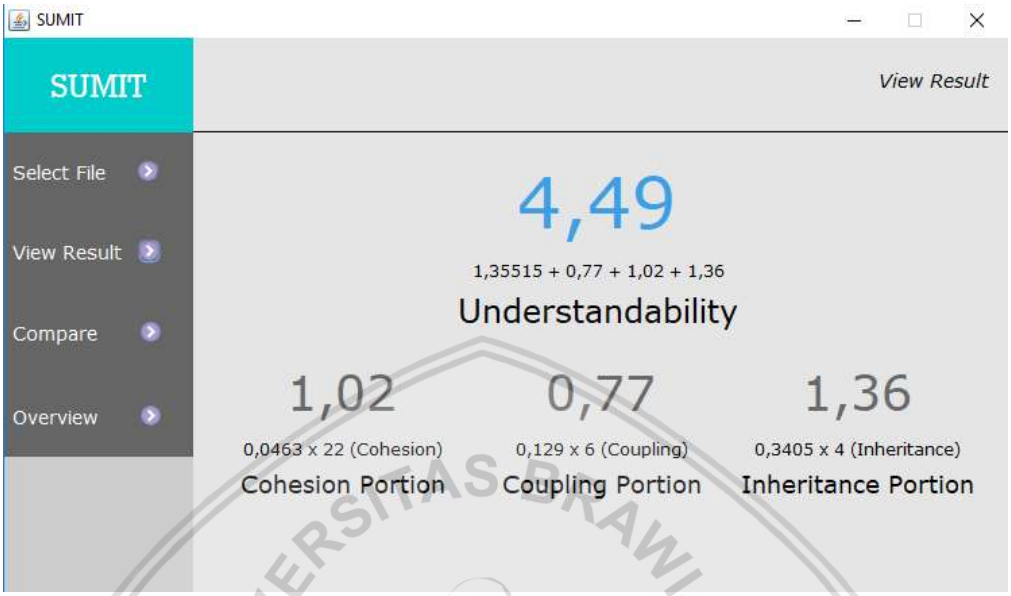
Berikut merupakan hasil implementasi dari aplikasi perangkat lunak SUMIT dan bagaimana kebutuhan fungsional aplikasi SUMIT antara lain seperti, *Browse* Berkas (SRS_F_SUMIT_001), Ukur Porsi *Coupling* (SRS_F_SUMIT_002), Ukur Porsi *Cohesion* (SRS_F_SUMIT_002), Ukur Porsi *Inheritance* (SRS_F_SUMIT_004), Ukur *Understandability* (SRS_F_SUMIT_005), Lihat Hasil (SRS_F_SUMIT_006), Lihat Ikhtisar (SRS_F_SUMIT_007), dan Lihat Banding *Design* (SRS_F_SUMIT_008) dijalankan.

Pada Gambar 5.6 adalah halaman seleksi berkas dan pengukuran yang merupakan hasil implementasi dari kebutuhan *Browse* berkas (SRS_SUMIT_001), kebutuhan Ukur Porsi *Coupling* (SRS_F_SUMIT_002), kebutuhan Ukur Porsi *Cohesion* (SRS_F_SUMIT_002), kebutuhan Ukur Porsi *Inheritance* (SRS_F_SUMIT_004), dan kebutuhan Ukur *Understandability* (SRS_F_SUMIT_005). Pada halaman seleksi berkas terdapat dua buah tombol yatu *browse* untuk menampilkan dialog pemilihan berkas dan tombol *process* untuk melakukan pengukuran ketika berkas telah berhasil dipilih.



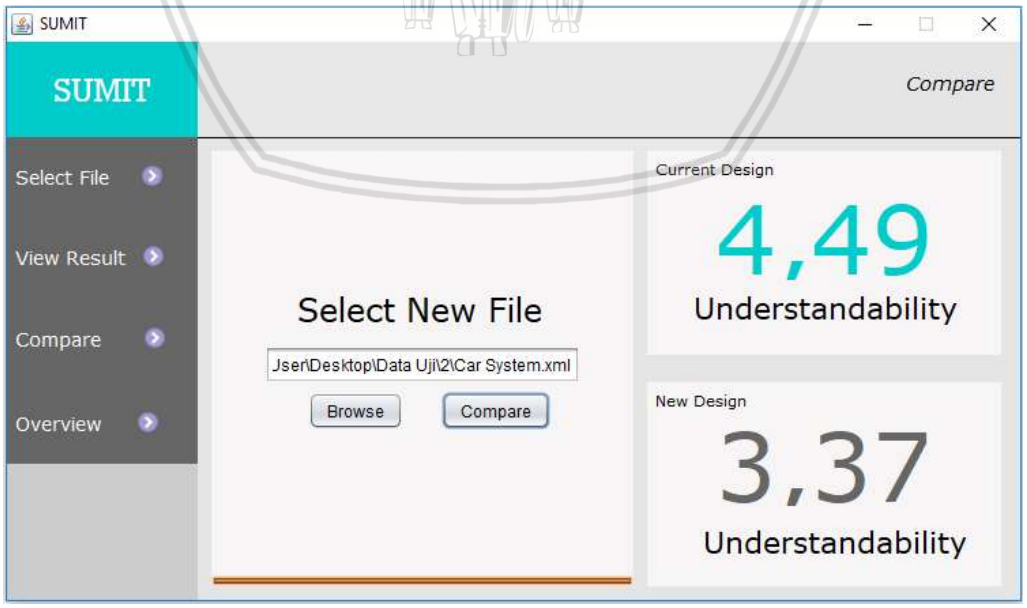
Gambar 5.6 Tampilan Halaman Seleksi Berkas dan Pengukuran

Pada Gambar 5.7 adalah halaman melihat hasil yang merupakan hasil implementasi dari kebutuhan Lihat Hasil (SRS_F_SUMIT_006). Pada halaman lihat hasil menampilkan hasil pengukuran *understandability* beserta rincian perhitunganya dan pengukuran porsi *cohesion, coupling, dan inheritance*.



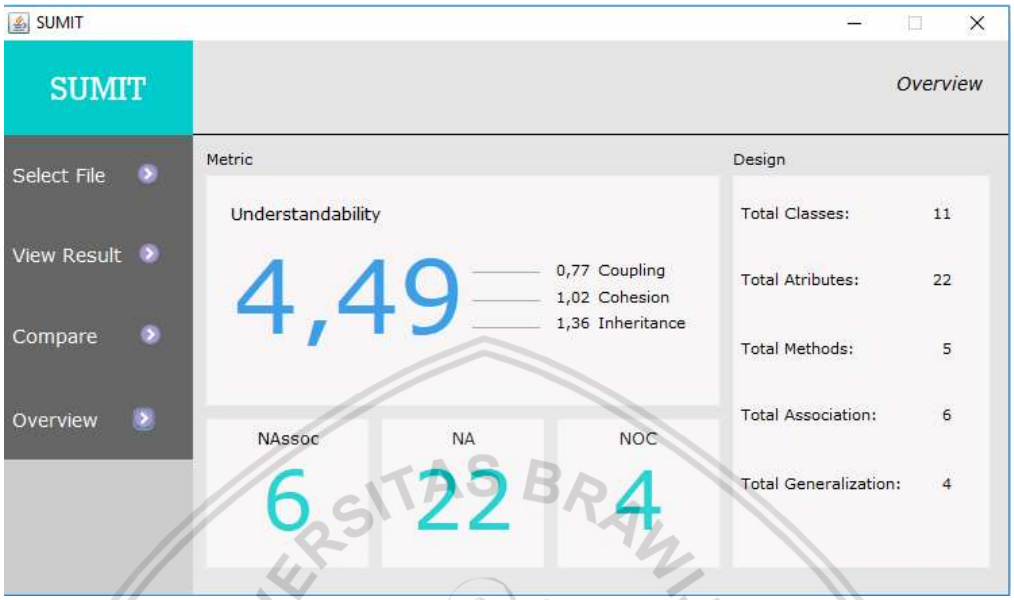
Gambar 5.7 Tampilan Halamn Lihat Hasil

Pada Gambar 5.8 adalah halaman melihat hasil yang merupakan hasil implementasi dari kebutuhan Banding *design* (SRS_F_SUMIT_008). Pada halaman membandingkan rancangan menyediakan tombol untuk memilih berkas rancangan baru untuk dibandingkan dengan rancangan yang sebelumnya telah berhasi dihitung. Kemudain tombol compare untuk menampilkan hasil perhitungan *understandability* diantara dua rancangan perangkat lunak.



Gambar 5.8 Tampilan Halaman Membandingkan Rancangan

Pada Gambar 5.9 adalah halaman lihat ikhtisar yang merupakan hasil implementasi dari kebutuhan Lihat Ikhtisar (SRS_F_SUMIT_007). Pada halaman ini SUMIT menampilkan hasil perhitungan *understandability*, nilai setiap metrik yang digunakan dan informasi dari rancangan yang terkait dengan *understandability*.



Gambar 5.9 Tampilan Halaman Lihat Ikhtisar

BAB 6 PENGUJIAN DAN ANALISIS HASIL

Pada bab ini penulis akan membahas mengenai proses pengujian terhadap aplikasi perangkat lunak SUMIT yang telah dikembangkan antara lain mengenai lingkungan pengujian perangkat lunak, identifikasi dan rencana pengujian, dan deskripsi dan hasil uji. Pengujian akan dilakukan melaui tiga macam pengujian yaitu pengujian unit, pengujian integrasi, dan pengujian validasi. Setelah pengujian selesai dilaksanakan penulis kemudian melakukan analisis hasil melihat korelasi dari nilai *understandabilty* yang berhasil diukur SUMIT terhadap nilai *maintainability*

6.1. Lingkungan Pengjujian Perangkat Lunak

6.1.1. Perangkat Lunak

Pada pengujian aplikasi perangkat lunak SUMIT digunakan beberapa *software* pendukung seperti pada tabel 6.1 dibawah ini:

Tabel 6.1 Perangkat Lunak Pengujian

Nama *Software*	Keterangan
NetBeans 8.2	Untuk menerima *input* manual menghitung *understandability* sebuah rancangan perangkat lunak
UML Diagram Project	Plugin pada NetBeans 8.2 untuk melakukan generate UML class diagram dari *project open source* yang didapat dari gitHub
PowerPoint	Membuat *flowgraph white box testing* untuk pengujian unit
Notepad	Untuk mencatat inputan manual dan menyimpan hasil perhitungan manual secara sementara dari setiap studi kasus

5.5.1. Hardware

Dalam melakukan pengujian aplikasi perangkat lunak SUMIT digunakan spesifikasi *hardware* seperti pada tabel 6.2 dibawah ini:

Tabel 6.2 Perangkat Keras Pengujian

Komponen *Hardware*	Keterangan
System Model	Asus Vivobook 14 X405UQ-BV567
Processor	Intel CORE i5 7th Generation 2.50GHz

Graphic Card	NVIDIA GEFORCE 940MX
Memory	4 GB
HDD	1TB

.

6.2. Pengujian Unit

Pada pengujian unit penulis mengambil tiga buah operasi utama antara lain operasi *getResult*(*double Coup, double Coh, double Inherit*) dari kelas *Understandability*, operasi *ValidateStruture(File xml)* dari kelas *SpecialXMLParser*, dan operasi SelectActionPerformed(ActionEvent evt) dari kelas *boundry Panel_Select*.

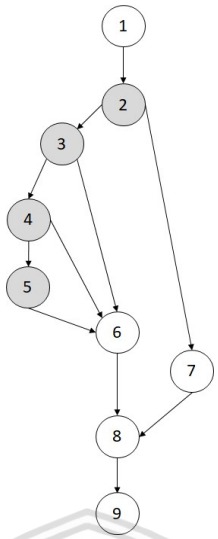
6.2.1. Pengujian Unit Operasi *getResult*

Berikut identifikasi jalur kasus uji secara *white box testing* untuk operasi *getResult()* dari kelas *Understandability* pada Gambar 6.1:

```
1  TYPE Cov is double;
   TYPE Coup is double;
   TYPE Coh is double;
   TYPE Inherit is double;
   TYPE understandability is double:
   Understandability = 0; } 2
                3      4          5
   IF Coup != 0 || Coh !=0 || Inherit != 0
     THEN understandability = Cov + Coup + Coh +Inherit; } 6
   ELSE understandability = 0 ; } 7
   ENDIF } 8
   RETURNVALUE understandability; } 9
   END
```

Gambar 6.1 Algoritme Operasi *getResult*

Berdasarkan Gambar 6.1 maka diperolehlah *flow graph* untuk operasi *getResult()* sebagai berikut pada Gambar 6.2:



Gambar 6.2 *Flow Graph* Operasi *getResult*

Berdasarkan Gambar 6.2 maka diperoleh hasil *cyclomatic complexity* sebagai berikut:

A. V(G) = 4 regions
B. V(G) = 10E – 8N + 2 = 4
C. V(G) = 3p + 1 = 4

Sehingga didapatkan 4 jalur independen (IP) untuk kasus uji antara lain adalah sebagai berikut:

A. Jalur 1: 1,2,7,8,9
B. Jalur 2: 1,2,3,6,8,9
C. Jalur 3: 1,2,3,4,6,8,9
D. Jalur 4: 1,2,3,4,5,6,8,9

Berdasarkan jalur independen yang telah diidentifikasi, maka kasus uji operasi getResult dapat didefinisikan pada Tabel sebagai berikut:

Tabel 6.3 Kasus Uji Operasi *getResult()*

No	Kasus Uji	Data *Input*	IP	Ekpektasi Hasil	Hasil Aktual	Status
1	Coup = 0	Coup = 0, Coh = 0, Inherit = 0	1,2,7,8,9	*Understandability* = 0	*Understandability* = 0	1
2	Coh = 0	Coup = 1, Coh = 0, Inherit = 0	1,2,3,6,8,9	*Understandability* = 2.33515	*Understandability* = 2.33515	1
3	Inherit = 0	Coup = 1, Coh = 1, Inherit = 0	1,2,3,4,6,8,9	*Understandability* = *Understandability* = 3.33515	*Understandability* = 3.33515	1
4	Coup > 0, Coh > 0, Inherit > 0	Coup = 1, Coh = 1 Inherit = 1	1,2,3,4,5,6,8,9	*Understandability* = *Understandability* = 4.33515	*Understandability* = 4.33515	1

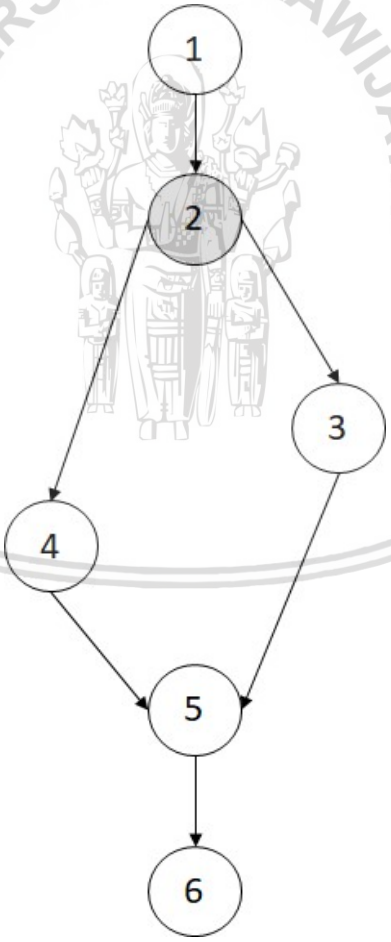
6.2.2. Pengujian Unit Operasi *SelectActionPerformed*

Berikut identifikasi jalur kasus uji secara *white box testing* untuk operasi *SelectActionPerformed(ActionEvent evt)* dari kelas *Understandability* pada Gambar 6.3:

```
1  Pb_Process Call setValue(argument: 0 TYPE String);
   TYPE returnVal is integer
   returnVal = FileChooser Call showOpenDialog(argument: this TYPE Object);
               2
   IF returnVal == JFileChooser Call APPROVE_OPTION THEN
             TYPE file is File;
             file = FileChooser Call getSelectedFile();
             txt_File Call setText(argument: file Call getAbsolutePath TYPE File);   } 3
   ELSE txt_File Call setText(argument: null TYPE String);
   FileChooser Call setSelectedFile(argument: null TYPE String);                     } 4
   ENDIF } 5
   END } 6
```

Gambar 6.3 Algoritme Operasi *SelectActionPerformed*

Berdasarkan Gambar 6.3 maka diperolehlah *flow graph* untuk operasi *SelectActionPerformed(ActionEvent evt)* sebagai berikut pada Gambar 6.4:



Gambar 6.4 *Flow Graph* Operasi *SelectActionPerformed(ActionEvent evt)*

Berdasarkan Gambar 6.4 maka diperoleh hasil *cyclomatic complexity* sebagai berikut:

A. V(G) = 2 regions

B. V(G) = 6E – 6N + 2 = 2

C. V(G) = 1p + 1 = 2

Sehingga didapatkan 2 jalur independen (IP) untuk kasus uji antara lain adalah sebagai berikut:

A. Jalur 1: 1,2,4,5,6

B. Jalur 2: 1,2,3,5,6

Berdasarkan jalur independen yang telah diidentifikasi, maka kasus uji operasi getResult dapat didefinisikan pada Tabel sebagai berikut:

Tabel 6.4 Kasus Uji Operasi *SelectActionPerformed(ActionEvent evt)*

No	Kasus Uji	Data *Input*	IP	Ekpektasi Hasil	Hasil Aktual	Status
1	returnVal != Approved	Tekan tombol *Cancel*	1,2,4,5,6	txt_file = null	txt_file = null	1
2	returnVal = Approved	Tekan tombol *Ok*	1,2,3,5,6	txt_file = *file's path*	txt_file = *file's path*	1

6.2.3. Pengujian Unit Operasi *SelectActionPerformed*

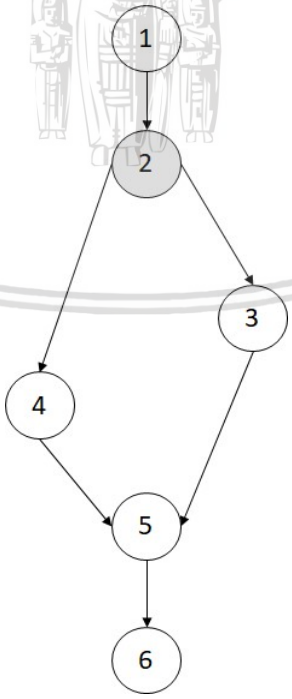
Berikut identifikasi jalur kasus uji secara *white box testing* untuk operasi *ValidateStructure (File xml)* dari kelas *Understandability* pada Gambar 6.5:

```
TYPE dbFactory is DocumentBuilderFactory;
TYPE dBuilderis DocumentBuilder;
TYPE doc is Document;
TYPE strc is NodeList;
TYPE nNode is Node;
TYPE eElement is Element;
dbFactory = DocumentBuilderFactory CALL newInstance();
dBuilder = dbFaktory CALL newDocumentBuilder();
doc = dBuilder CALL parse(argument: xml TYPE File);
doc CALL getDocumentElement() CALL normalize();
doc CALL getDocumentElement();
strc = doc CALL getElementsByTagName("Project");
nNode = strc CALL item(argument: 0);
eElement = (Element) nNode;

IF (!"simple" = eElement CALL getAttribute(argument: Xml_structure TYPE String))
      THEN valid = false;
ELSE valid = true;
ENDIF;
return valid;
END
```

Gambar 6.5 Algoritme Operasi *ValidateStructure*

Berdasarkan Gambar 6.5 maka diperolehlah *flow graph* untuk operasi *SelectActionPerformed(ActionEvent evt)* sebagai berikut pada Gambar 6.6:



Gambar 6.6 *Flow Graph* Operasi *ValidateStructure (File xml)*

Berdasarkan Gambar 6.4 maka diperoleh hasil *cyclomatic complexity* sebagai berikut:

D. V(G) = 2 regions

E. V(G) = 6E – 6N + 2 = 2

F. V(G) = 1p + 1 = 2

Sehingga didapatkan 2 jalur independen (IP) untuk kasus uji antara lain adalah sebagai berikut:

C. Jalur 1: 1,2,4,5,6

D. Jalur 2: 1,2,3,5,6

Berdasarkan jalur independen yang telah diidentifikasi, maka kasus uji operasi getResult dapat didefinisikan pada Tabel 6.5 sebagai berikut:

Tabel 6.5 Kasus Uji Operasi *getResult()*

No	Kasus Uji	Data *Input*	IP	Ekpektasi Hasil	Hasil Aktual	Status
1	!"Simple" = true	Berkas XML dengan *traditional structure*	1,2,4,5,6	valid = false	valid = false	1
2	!"Simple" = false	Berkas XML dengan *simplel structure*	1,2,3,5,6	valid = true	valid = true	1

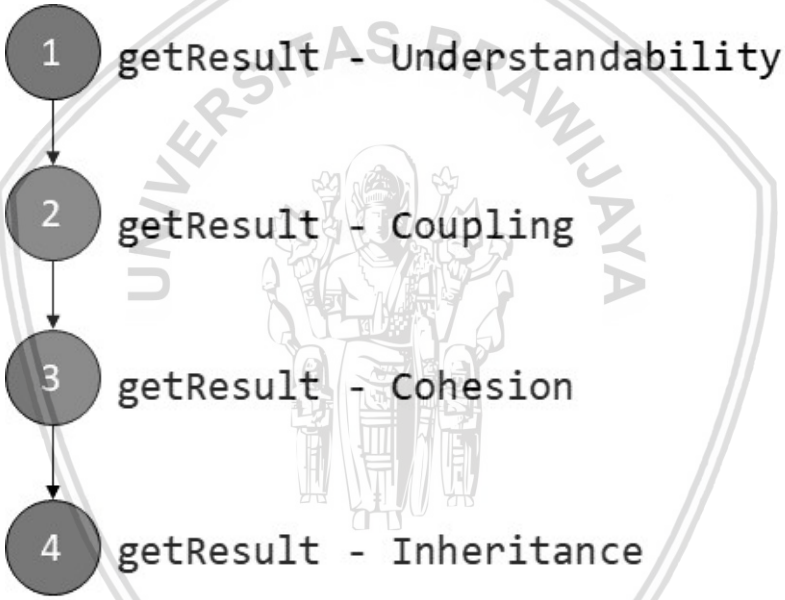
6.3. Pengujian Integrasi

Pengujian integrasi adalah pengujian yang difokuskan pada gabungan unit-unit yang membentuk kesatuan fungsional. Pengujian integrasi dilakukan dengan menguji interaksi antar kelas menggunakan pendekatan *top-down testing* yang meliputi, kelas *control Coupling*, kelas *control Coupling*, kelas *control Coupling*, dan kelas *control Understandability* dalam menjalankan fungsionalitas ukur *understandability*. Pada Tabel 6.6 menunjukan identifikasi kelas/module yang akan diuji sebagai berikut:

Tabel 6.6 Identifikasi Kelas

Operasi yang Terlibat	Kelas yang Terlibat	Tujuan
getResult(int Asso)	*Coupling*	Mengukur nilai *understandability* berdasarkan masukkan dari pengguna
getResult(int Attr)	*Cohesion*	
getResult(int Child)	*Inheritance*	
getResult(double Coup, double Coh, double Inherit)	*Understandability*	

Berdasarkan Tabel 6.6 dengan menggunakan pedekatan *top-down testing* maka diperoleh model pengujian integrasi seperti pada Gambar 6.7 sebagai berikut:



Gambar 6.7 *Top-Down Testing*

Dari *top-down testing* yang sesuai pada gambar 6.7 menunjukan hubungan *node* 1 ke *node* 2 yaitu operasi *getResult* pada *Coupling* akan memberikan hasil porsi *coupling* yang akan menjadi masukan pada parameter *Coup* pada operasi *getResult Understandability*. Kemudian hubungan *node* 1-2 ke *node* 3 yaitu operasi *getResult* pada *Cohesion* akan memberikan hasil porsi *cohesion* yang akan menjadi masukan pada parameter *Coh* pada operasi *getResult Understandability*. Dan hubungan *node* 1-2-3 ke *node* 4 yaitu operasi *getResult* pada *Inheritance* akan memberikan hasil porsi *inheritance* yang akan menjadi masukan pada parameter *Inherit* pada operasi *getResult Understandability*. Maka diperoleh langkah pengujian integrasi sebagai berikut yang dinyatakan pada Tabel 6.7:

Tabel 6.7 Langkah Uji Pengujian Integrasi

No	Langkah Uji	Keterangan
1	Node 1 + Node 2	*getResult(int Asso)* dari Coupling dipanggil oleh *getResult(double Coup, double Coh, double Inherit*) dari *Understandability*
2	(Node 1 + Node 2)+ Node 3	*getResult(int Attr)* dari *Cohesion* dipanggil oleh *getResult*(*double Coup, double Coh, double Inherit*) dari *Understandability* yang sudah terdapat nilai *getResult(int Asso)* dari *Coupling*
3	(Node 1 + Node 2+ Node 3) + Node 4	*getResult(int Child)* dari *Inheritance* dipanggil oleh *getResult*(*double Coup, double Coh, double Inherit*) dari *Understandability* yang sudah terdapat nilai *getResult(int Asso)* dari *Coupling* dan nilai *getResult(int Attr)* dari *Cohesion*

Dari langkah pada Tabel 6.7 maka dibuatlah *stub*/modul pengganti didalam operasi *getResult* pada *Understandability*. Berikut ini pada Tabel 6.8 dilakukan pengunaan *stub* mencoba langkah 1.

Tabel 6.8 Menjalankan Integrasi Langkah 1

```
public class Understandability{
    private final double Cov = 1.33515;

    public double getResult(double Coup, double Coh, double
Inherit) {
        double understandability;

        if (Coup != 0 || Coh != 0 || Inherit != 0) {
            understandability = Cov + Coup + Coh + Inherit;
        } else {
            understandability = 0;
        }
        return understandability;
    }}
public class cobastub{
    public static void main(String[] args) {
        Understandability x = new Understandability();
        double stubCoup = 1; //stub untuk mencoba integrasi
        //langkah 1

        System.out.println(x.getResult(stubCoup, 0, 0));
    }}
```

```
Output - Skripsi_SUMIT (run)
run:
2.33515
BUILD SUCCESSFUL (total time: 0 seconds)
```

Hasil dari langkah 1 yang ditunjukan pada Tabel 6.8 ketika menggunakan *stubCoup* tidak menghasilkan hasil 0 pada *getResult Understandability* yang berarti integrasi berhasil.

Dari langkah pada Tabel 6.7 maka dibuatlah *stub*/modul pengganti didalam operasi *getResult* pada *Understandability*. Berikut ini pada Tabel 6.9 dilakukan pengunaan *stub* mencoba langkah 2.

Tabel 6.9 Menjalankan Integrasi Langkah 2

```
public class Understandability{
    private final double Cov = 1.33515;

    public double getResult(double Coup, double Coh, double
Inherit) {
        double understandability;

        if (Coup != 0 || Coh != 0 || Inherit != 0) {
            understandability = Cov + Coup + Coh + Inherit;
        } else {
            understandability = 0;
        }
        return understandability;
    }}
public class cobastub{
    public static void main(String[] args) {
        Understandability x = new Understandability();
        double stubCoup = 1;
        double stubCoh = 1;// stub untuk mencoba integrasi
        //langkah 2

        System.out.println(x.getResult(stubCoup, stubCoh, 0));
    }}
```

```
Output - Skripsi_SUMIT (run)
run:
3.33515
BUILD SUCCESSFUL (total time: 0 seconds)
```

Hasil dari langkah 2 yang ditunjukan pada Tabel 6.9 ketika menggunakan *stubCoup* dan *stubCoh* menghasilkan hasil 3.33515 pada *getResult Understandability* yang berarti integrasi berhasil.

Dari langkah pada Tabel 6.7 maka dibuatlah *stub*/modul pengganti didalam operasi *getResult* pada *Understandability*. Berikut ini pada Tabel 6.10 dilakukan pengunaan *stub* mencoba langkah 3.

Tabel 6.10 Menjalankan Integrasi Langkah 3

```
public class Understandability{
    private final double Cov = 1.33515;

    public double getResult(double Coup, double Coh, double
Inherit) {
        double understandability;
```

```
        if (Coup != 0 || Coh != 0 || Inherit != 0) {
            understandability = Cov + Coup + Coh + Inherit;
        } else {
            understandability = 0;
        }
        return understandability;
    }}
public class cobastub{
    public static void main(String[] args) {
        Understandability x = new Understandability();
        double stubCoup = 1;
        double stubCoh = 1;
        double stubInherit = 1;// stub untuk mencoba integrasi
        //langkah 3

        System.out.println(x.getResult(stubCoup,         stubCoh,
stubInherit));
    }}
```

```
Output - Skripsi_SUMIT (run)
run:
4.3351500000000005
BUILD SUCCESSFUL (total time: 0 seconds)
```

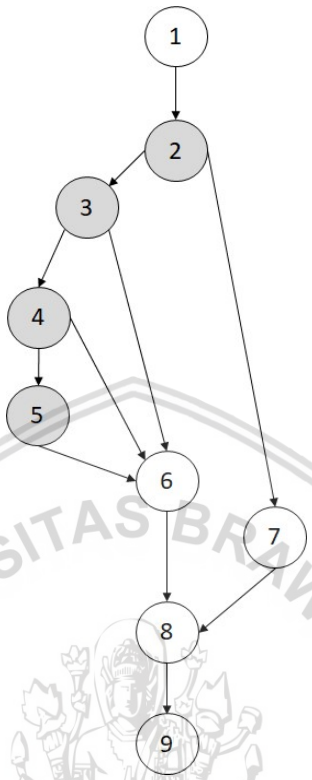
Hasil dari langkah 3 yang ditunjukan pada Tabel 6.10 ketika menggunakan *stubCoup*, *stubCoh*, dan *stubInherit* menghasilkan hasil 4.335150000000005 pada *getResult Understandability* yang berarti integrasi berhasil.

Dari hasil yang diberikan dengan menggunakan *stub* secara keseluruhan akan membentuk algoritme berikut sesuai Gambar 6.8.

```
1 { TYPE Cov is double;
    TYPE Coup is double;
    TYPE Coh is double;
    TYPE Inherit is double;
    TYPE understandability is double:
    Understandability = 0; } 2
    IF Coup != 0 || Coh !=0 || Inherit != 0
       THEN understandability = Cov + Coup + Coh +Inherit; } 6
    ELSE understandability = 0 ; } 7
    ENDIF } 8
    RETURNVALUE understandability; } 9
    END
```

Gambar 6.8 Algoritme integrasi *getResult* Kelas *Understandability*

Berdasarkan algoritme pada Gambar 6.8 maka diperolehlah *flow graph* untuk operasi *getResult(double Coup, double Coh, double Inherit)* hasil integrasi sebagai berikut pada Gambar 6.9.



Gambar 6.9 *Flow Graph* Pengujian integrasi *getResult* Kelas *Understandability*

Berdasarkan Gambar 6.9 maka diperoleh hasil *cyclomatic complexity* sebagai berikut:

A. V(G) = 4 regions
B. V(G) = 10E – 8N + 2 = 4
C. V(G) = 3p + 1 = 4

Sehingga didapatkan 4 jalur independen (IP) untuk kasus uji antara lain adalah sebagai berikut:

A. Jalur 1: 1,2,7,8,9
B. Jalur 2: 1,2,3,6,8,9
C. Jalur 3: 1,2,3,4,6,8,9
D. Jalur 4: 1,2,3,4,5,6,8,9

Berdasarkan jalur independen yang telah diidentifikasi, maka kasus uji operasi getResult dapat didefinisikan pada Tabel sebagai berikut:

Tabel 6.11 Kasus Uji Intgrasi Operasi getResult

No	Kasus Uji	Data *Input*	IP	Ekpektasi Hasil	Hasil Aktual	Status
1	Coup = 0	Coup = 0, Coh = 0, Inherit = 0	1,2,7,8,9	*Understandability* = 0	*Understandability* = 0	1
2	Coh = 0	Coup = 1, Coh = 0, Inherit = 0	1,2,3,6,8,9	*Understandability* = 2.33515	*Understandability* = 2.33515	1
3	Inherit = 0	Coup = 1, Coh = 1, Inherit = 0	1,2,3,4,6,8,9	*Understandability* = *Understandability* = 3.33515	*Understandability* = 3.33515	1
4	Coup > 0, Coh > 0, Inherit > 0	Coup = 1, Coh = 1 Inherit = 1	1,2,3,4,5,6,8,9	*Understandability* = *Understandability* = 4.33515	*Understandability* = 4.33515	1

6.4. Pengujian Validasi

Pada pengujian validasi menggunakan pengujian *black box* untuk mengetahui apakah sistem yang dibangun sudah benar sesuai dengan yang dibutuhkan baik untuk kebutuhan fungsional dan non fungsional. Untuk pengujian validasi kebutuhan fungsional terdapat delapan buah kebutuhan yaitu, *browse* berkas, ukur *understandability*, ukur porsi *coupling*, ukur porsi *cohesion*, ukur porsi *inheritance*, lihat hasil, lihat ikhtisar dan banding *design* yang akan diuji berdasarkan spesifikasi kebutuhan dibandingkan bersama hasil aktual dari aplikasi perangkat lunak SUMIT untuk melihat apakah hasil aktual dapat memenuhi target pada spesifikasi kebutuhan fungsional. Sedangkan untuk kebutuhan non-fungsional terdapat satu buah kebutuhan yaitu efisiensi yang akan diuji dengan cara sesuai pada persaman (2) dan persamaan (3).

6.4.1. Pengujian Validasi *Browse* Berkas

Berikut ini merupakan pengujuan validasi untuk kebutuhan *browse* berkas. Pada Tabel 6.12 merupakan hasil pengujian validasi *main flow Browse* Berkas dan pada Tabel 6.13 merupakan hasil pengujian validasi *alternative flow* 1 *Browse* Berkas.

Tabel 6.12 Pengujian Validasi *Main flow Browse* Berkas

Kode SRS	SRS_F_SUMIT_001
Use case	*Browse* Berkas
Tujuan Pengujian	Memastikan SUMIT dapat memenuhi jalur utama kebutuhan fungsional *Browse* Berkas

Data Input	Berkas XML atau non-XML
Prosedur Pengujian	1. Pengguna membuka halaman seleksi berkas 2. Pengguna menekan tombol *browse* 3. Pengguna memilih sebuah berkas 4. Pengguna menekan tombol *open*
Expected Result	SUMIT mengisi text field berkas dengan lokasi keberadaan berkas
Actual Result	SUMIT mengisi text field berkas dengan lokasi keberadaan berkas
Status	Valid

Tabel 6.13 Pengujian Validasi *Alternative Flow* 1 *Browse* Berkas

Kode SRS	SRS_F_SKRIPSI_001
Use case	*Browse* Berkas
Tujuan Pengujian	Memastikan SUMIT dapat memenuhi kondisi alternatif kebutuhan fungsional *Browse* Berkas
Data Input	-
Prosedur Pengujian	1. Pengguna membuka halaman seleksi berkas 2. Pengguna menekan tombol *browse* 3. Pengguna memilih sebuah berkas 4. Pengguna menekan tombol *cancel*
Expected Result	SUMIT membiarkan text field berkas bernilai kosong
Actual Result	SUMIT mengisi text field berkas dengan lokasi keberadaan berkas
Status	Valid

6.4.2. Pengujian Validasi Ukur Porsi *Coupling*

Berikut ini merupakan pengujuan validasi untuk kebutuhan Ukur Porsi *Coupling*. Pada Tabel 6.14 merupakan hasil pengujian validasi *main flow* Ukur Porsi *Coupling*, pada Tabel 6.15 merupakan hasil pengujian validasi *alternative flow* 1 Ukur Porsi *Coupling*, pada Tabel 6.16 merupakan hasil pengujian validasi *alternative flow* 2 Ukur Porsi *Coupling*, dan pada pada Tabel 6.17 merupakan hasil pengujian validasi *alternative flow* 3 Ukur Porsi *Coupling*.

Tabel 6.14 Pengujian Validasi *Main Flow* Ukur Porsi *Coupling*

Kode SRS	SRS_F_SUMIT_002
Use case	Ukur Porsi *Coupling*

Tujuan Pengujian	Memastikan SUMIT dapat memenuhi jalur utama kebutuhan fungsional Ukur Porsi *Coupling*
Data Input	Berkas XML dengan struktur *simple*
Prosedur Pengujian	1. Pengguna membuka halaman seleksi berkas 2. Pengguna menekan tombol *browse* 3. Pengguna memilih sebuah berkas XML rancangan perangkat lunak berstruktur simple 4. Pengguna menekan tombol *open* 5. Pengguna menekan tombol *process*
Expected Result	SUMIT berhasil mengukur berkas XML rancangan perangkat lunak
Actual Result	SUMIT berhasil mengukur berkas XML rancangan perangkat lunak
Status	Valid

Tabel 6.15 Pengujian Validasi *Alternative Flow* 1 Ukur Porsi *Coupling*

Kode SRS	SRS_F_SUMIT_002
Use case	Ukur Porsi *Coupling*
Tujuan Pengujian	Memastikan SUMIT dapat memenuhi jalur alternatif pertama kebutuhan fungsional Ukur Porsi *Coupling*
Data Input	Berkas non-XML
Prosedur Pengujian	1. Pengguna membuka halaman seleksi berkas 2. Pengguna menekan tombol *browse* 3. Pengguna memilih sebuah berkas non-XML rancangan perangkat lunak berstruktur simple 4. Pengguna menekan tombol *open* 5. Pengguna menekan tombol *process*
Expected Result	SUMIT menampilkan pesan berkas yang dipilih tidak menggunakan *format* XML
Actual Result	SUMIT menampilkan pesan berkas yang dipilih tidak menggunakan *format* XML
Status	Valid

Tabel 6.16 Pengujian Validasi *Alternative Flow* 2 Ukur Porsi *Coupling*

Kode SRS	SRS_F_SUMIT_002
Use case	Ukur Porsi *Coupling*

Tujuan Pengujian	Memastikan SUMIT dapat memenuhi jalur alternatif kedua kebutuhan fungsional Ukur Porsi *Coupling*
Data Input	Berkas XML tidak mengunakan struktur *simple*
Prosedur Pengujian	1. Pengguna membuka halaman seleksi berkas 2. Pengguna menekan tombol *browse* 3. Pengguna memilih sebuah berkas XML rancangan perangkat lunak berstruktur non-simple 4. Pengguna menekan tombol *open* 5. Pengguna menekan tombol *process*
Expected Result	SUMIT menampilkan pesan berkas XML yang dipilih tidak menggunakan struktur simple
Actual Result	SUMIT menampilkan pesan berkas XML yang dipilih tidak menggunakan struktur simple
Status	Valid

Tabel 6.17 Pengujian Validasi *Alternative Flow* 3 Ukur Porsi *Coupling*

Kode SRS	SRS_F_SUMIT_002
Use case	Ukur Porsi *Coupling*
Tujuan Pengujian	Memastikan SUMIT dapat memenuhi jalur alternatif ketiga kebutuhan fungsional Ukur Porsi *Coupling*
Data Input	-
Prosedur Pengujian	1. Pengguna membuka halaman seleksi berkas 2. Pengguna menekan tombol *process*
Expected Result	SUMIT menampilkan pesan tidak ada berkas yang dipilih
Actual Result	SUMIT menampilkan pesan tidak ada berkas yang dipilih
Status	Valid

6.4.3. Pengujian Validasi Ukur Porsi *Cohesion*

Berikut ini merupakan pengujuan validasi untuk kebutuhan Ukur Porsi *Cohesion*. Pada Tabel 6.18 mer`upakan hasil pengujian validasi *main flow* Ukur Porsi *Cohesion*, pada Tabel 6.19 merupakan hasil pengujian validasi *alternative flow* 1 Ukur Porsi *Cohesion*, pada Tabel 6.20 merupakan hasil pengujian validasi *alternative flow* 2 Ukur Porsi *Cohesion*, dan pada pada Tabel 6.21 merupakan hasil pengujian validasi *alternative flow* 3 Ukur Porsi *Cohesion*.

Tabel 6.18 Pengujian Validasi *Main Flow* Ukur Porsi *Cohesion*

Kode SRS	SRS_F_SUMIT_003

Use case	Ukur Porsi *Cohesion*
Tujuan Pengujian	Memastikan SUMIT dapat memenuhi jalur utama kebutuhan fungsional Ukur Porsi *Cohesion*
Data Input	Berkas XML dengan struktur *simple*
Prosedur Pengujian	1. Pengguna membuka halaman seleksi berkas 2. Pengguna menekan tombol *browse* 3. Pengguna memilih sebuah berkas XML rancangan perangkat lunak berstruktur simple 4. Pengguna menekan tombol *open* 5. Pengguna menekan tombol *process*
Expected Result	SUMIT berhasil mengukur berkas XML rancangan perangkat lunak
Actual Result	SUMIT berhasil mengukur berkas XML rancangan perangkat lunak
Status	Valid

Tabel 6.19 Pengujian Validasi *Alternative Flow* 1 Ukur Porsi *Cohesion*

Kode SRS	SRS_F_SUMIT_003
Use case	Ukur Porsi *Cohesion*
Tujuan Pengujian	Memastikan SUMIT dapat memenuhi jalur alternatif pertama kebutuhan fungsional Ukur Porsi *Cohesion*
Data Input	Berkas non-XML
Prosedur Pengujian	1. Pengguna membuka halaman seleksi berkas 2. Pengguna menekan tombol *browse* 3. Pengguna memilih sebuah berkas non-XML rancangan perangkat lunak berstruktur simple 4. Pengguna menekan tombol *open* 5. Pengguna menekan tombol *process*
Expected Result	SUMIT menampilkan pesan berkas yang dipilih tidak menggunakan *format* XML
Actual Result	SUMIT menampilkan pesan berkas yang dipilih tidak menggunakan *format* XML
Status	Valid

Tabel 6.20 Pengujian Validasi *Alternative Flow* 2 Ukur Porsi *Cohesion*

Kode SRS	SRS_F_SUMIT_003
Use case	Ukur Porsi *Cohesion*

Tujuan Pengujian	Memastikan SUMIT dapat memenuhi jalur alternatif kedua kebutuhan fungsional Ukur Porsi *Cohesion*
Data Input	Berkas XML tidak mengunakan struktur *simple*
Prosedur Pengujian	1. Pengguna membuka halaman seleksi berkas 2. Pengguna menekan tombol *browse* 3. Pengguna memilih sebuah berkas XML rancangan perangkat lunak berstruktur non-simple 4. Pengguna menekan tombol *open* 5. Pengguna menekan tombol *process*
Expected Result	SUMIT menampilkan pesan berkas XML yang dipilih tidak menggunakan struktur simple
Actual Result	SUMIT menampilkan pesan berkas XML yang dipilih tidak menggunakan struktur simple
Status	Valid

Tabel 6.21 Pengujian Validasi *Alternative Flow* 3 Ukur Porsi *Cohesion*

Kode SRS	SRS_F_SUMIT_003
Use case	Ukur Porsi *Cohesion*
Tujuan Pengujian	Memastikan SUMIT dapat memenuhi jalur alternatif ketiga kebutuhan fungsional Ukur Porsi *Cohesion*
Data Input	-
Prosedur Pengujian	1. Pengguna membuka halaman seleksi berkas 2. Pengguna menekan tombol *process*
Expected Result	SUMIT menampilkan pesan tidak ada berkas yang dipilih
Actual Result	SUMIT menampilkan pesan tidak ada berkas yang dipilih
Status	Valid

6.4.4. Pengujian Validasi Ukur Porsi *Inheritance*

Berikut ini merupakan pengujuan validasi untuk kebutuhan Ukur Porsi *Inheritance*. Pada Tabel 6.22 merupakan hasil pengujian validasi *main flow* Ukur Porsi *Inheritance*, pada Tabel 6.23 merupakan hasil pengujian validasi *alternative flow* 1 Ukur Porsi *Inheritance*, pada Tabel 6.24 merupakan hasil pengujian validasi *alternative flow* 2 Ukur Porsi *Inheritance*, dan pada pada Tabel 6.25 merupakan hasil pengujian validasi *alternative flow* 3 Ukur Porsi *Inheritance*.

Tabel 6.22 Pengujian Validasi *Main Flow* Ukur Porsi *Inheritance*

Kode SRS	SRS_F_SUMIT_004

Use case	Ukur Porsi *Inheritance*
Tujuan Pengujian	Memastikan SUMIT dapat memenuhi jalur utama kebutuhan fungsional Ukur Porsi *Inheritance*
Data Input	Berkas XML dengan struktur *simple*
Prosedur Pengujian	1. Pengguna membuka halaman seleksi berkas 2. Pengguna menekan tombol *browse* 3. Pengguna memilih sebuah berkas XML rancangan perangkat lunak berstruktur *simple* 4. Pengguna menekan tombol *open* 5. Pengguna menekan tombol *process*
Expected Result	SUMIT berhasil mengukur berkas XML rancangan perangkat lunak
Actual Result	SUMIT berhasil mengukur berkas XML rancangan perangkat lunak
Status	Valid

Tabel 6.23 Pengujian Validasi *Alternative Flow* 1 Ukur Porsi *Inheritance*

Kode SRS	SRS_F_SUMIT_004
Use case	Ukur Porsi *Inheritance*
Tujuan Pengujian	Memastikan SUMIT dapat memenuhi jalur alternatiff pertama kebutuhan fungsional Ukur Porsi *Inheritance*
Data Input	Berkas non-XML
Prosedur Pengujian	1. Pengguna membuka halaman seleksi berkas 2. Pengguna menekan tombol *browse* 3. Pengguna memilih sebuah berkas non-XML rancangan perangkat lunak berstruktur *simple* 4. Pengguna menekan tombol *open* 5. Pengguna menekan tombol *process*
Expected Result	SUMIT menampilkan pesan berkas yang dipilih tidak menggunakan *format* XML
Actual Result	SUMIT menampilkan pesan berkas yang dipilih tidak menggunakan *format* XML
Status	Valid

Tabel 6.24 Pengujian Validasi *Alternative Flow* 2 Ukur Porsi *Inheritance*

Kode SRS	SRS_F_SUMIT_004
Use case	Ukur Porsi *Inheritance*

Tujuan Pengujian	Memastikan SUMIT dapat memenuhi jalur alternatiff kedua kebutuhan fungsional Ukur Porsi *Inheritance*
Data Input	Berkas XML tidak mengunakan struktur simple
Prosedur Pengujian	1. Pengguna membuka halaman seleksi berkas 2. Pengguna menekan tombol *browse* 3. Pengguna memilih sebuah berkas XML rancangan perangkat lunak berstruktur non-simple 4. Pengguna menekan tombol *open* 5. Pengguna menekan tombol *process*
Expected Result	SUMIT menampilkan pesan berkas XML yang dipilih tidak menggunakan struktur simple
Actual Result	SUMIT menampilkan pesan berkas XML yang dipilih tidak menggunakan struktur simple
Status	Valid

Tabel 6.25 Pengujian Validasi *Alternative Flow* 3 Ukur Porsi *Inheritance*

Kode SRS	SRS_F_SUMIT_004
Use case	Ukur Porsi *Inheritance*
Tujuan Pengujian	Memastikan SUMIT dapat memenuhi jalur alternatif ketiga kebutuhan fungsional Ukur Porsi *Inheritance*
Data Input	-
Prosedur Pengujian	1. Pengguna membuka halaman seleksi berkas 2. Pengguna menekan tombol *process*
Expected Result	SUMIT menampilkan pesan tidak ada berkas yang dipilih
Actual Result	SUMIT menampilkan pesan tidak ada berkas yang dipilih
Status	Valid

6.4.5. Pengujian Validasi Ukur *Understandability*

Berikut ini merupakan pengujuan validasi untuk kebutuhan Ukur *Understandability*. Pada Tabel 6.26 merupakan hasil pengujian validasi *main flow* Ukur *Understandability*, pada Tabel 6.27 merupakan hasil pengujian validasi *alternative flow* 1 Ukur *Understandability*, pada Tabel 6.28 merupakan hasil pengujian validasi *alternative flow* 2 Ukur *Understandability*, dan pada pada Tabel 6.29 merupakan hasil pengujian validasi *alternative flow* 3 Ukur *Understandability*.

Tabel 6.26 Pengujian Validasi *Main Flow* Ukur *Understandability*

Kode SRS	SRS_F_SUMIT_005

Use case	Ukur *Understandability*
Tujuan Pengujian	Memastikan SUMIT dapat memenuhi jalur utama kebutuhan fungsional Ukur *Understandability*
Data Input	Berkas XML dengan struktur *simple*
Prosedur Pengujian	1. Pengguna membuka halaman seleksi berkas 2. Pengguna menekan tombol *browse* 3. Pengguna memilih sebuah berkas XML rancangan perangkat lunak berstruktur *simple* 4. Pengguna menekan tombol *open* 5. Pengguna menekan tombol *process*
Expected Result	SUMIT berhasil mengukur berkas XML rancangan perangkat lunak
Actual Result	SUMIT berhasil mengukur berkas XML rancangan perangkat lunak
Status	Valid

Tabel 6.27 Pengujian Validasi *Alternative Flow* 1 Ukur *Understandability*

Kode SRS	SRS_F_SUMIT_005
Use case	Ukur *Understandability*
Tujuan Pengujian	Memastikan SUMIT dapat memenuhi jalur alternatiff pertama kebutuhan fungsional Ukur *Understandability*
Data Input	Berkas non-XML
Prosedur Pengujian	1. Pengguna membuka halaman seleksi berkas 2. Pengguna menekan tombol *browse* 3. Pengguna memilih sebuah berkas non-XML rancangan perangkat lunak berstruktur simple 4. Pengguna menekan tombol *open* 5. Pengguna menekan tombol *process*
Expected Result	SUMIT menampilkan pesan berkas yang dipilih tidak menggunakan *format* XML
Actual Result	SUMIT menampilkan pesan berkas yang dipilih tidak menggunakan *format* XML
Status	Valid

Tabel 6.28 Pengujian Validasi *Alternative Flow* 2 Ukur *Understandability*

Kode SRS	SRS_F_SUMIT_005
Use case	Ukur *Understandability*

Tujuan Pengujian	Memastikan SUMIT dapat memenuhi jalur alternatiff kedua kebutuhan fungsional Ukur *Understandability*
Data Input	Berkas XML tidak mengunakan struktur *simple*
Prosedur Pengujian	1. Pengguna membuka halaman seleksi berkas 2. Pengguna menekan tombol *browse* 3. Pengguna memilih sebuah berkas XML rancangan perangkat lunak berstruktur non-*simple* 4. Pengguna menekan tombol *open* 5. Pengguna menekan tombol *process*
Expected Result	SUMIT menampilkan pesan berkas XML yang dipilih tidak menggunakan struktur simple
Actual Result	SUMIT menampilkan pesan berkas XML yang dipilih tidak menggunakan struktur simple
Status	Valid

Tabel 6.29 Pengujian Validasi *Alternative Flow* 3 Ukur *Understandability*

Kode SRS	SRS_F_SUMIT_005
Use case	Ukur *Understandability*
Tujuan Pengujian	Memastikan SUMIT dapat memenuhi jalur alternatif ketiga kebutuhan fungsional Ukur *Understandability*
Data Input	-
Prosedur Pengujian	1. Pengguna membuka halaman seleksi berkas 2. Pengguna menekan tombol *process*
Expected Result	SUMIT menampilkan pesan tidak ada berkas yang dipilih
Actual Result	SUMIT menampilkan pesan tidak ada berkas yang dipilih
Status	Valid

6.4.6. Pengujian Validasi Lihat Hasil

Berikut ini merupakan pengujuan validasi untuk kebutuhan Lihat Hasil. Pada Tabel 6.30 merupakan hasil pengujian validasi *main flow* Lihat Hasil.

Tabel 6.30 Pengujian Validasi *Main Flow* Lihat Hasil

Kode SRS	SRS_F_SUMIT_006
Use case	Lihat Hasil
Tujuan Pengujian	Memastikan SUMIT dapat memenuhi jalur utama kebutuhan fungsional Lihat Hasil
Data Input	-
Prosedur Pengujian	1. Pengguna membuka halaman lihat hasil
Expected Result	SUMIT menampilkan nilai hasil pengukuran *understandability*, porsi *coupling*, porsi *cohesion*, porsi *inheritance*, dan perincian pengukuran *multivariate understandability metric*
Actual Result	SUMIT menampilkan nilai hasil pengukuran *understandability*, porsi *coupling*, porsi *cohesion*, porsi *inheritance*, dan perincian pengukuran *multivariate understandability metric*
Status	Valid

6.4.7. Pengujian Validasi Lihat Ikhtisar

Berikut ini merupakan pengujuan validasi untuk kebutuhan Lihat Ikhtisar. Pada Tabel 6.31 merupakan hasil pengujian validasi *main flow* Lihat Ikhtisar.

Tabel 6.31 Pengujian Validasi *Main Flow* Lihat Ikhtisar

Kode SRS	SRS_F_SUMIT_007
Use case	Lihat Ikhtisar
Tujuan Pengujian	Memastikan SUMIT dapat memenuhi jalur utama kebutuhan fungsional Lihat Ikhtisar
Data Input	-
Prosedur Pengujian	1. Pengguna membuka halaman lihat ikhtisar
Expected Result	SUMIT menampilkan hasil ikhtisar berupa hasil pengukuran *understandability*, porsi *coupling*, porsi *cohesion*, porsi *inheritance*, nilai metrik NAssoc, metrik NA, metrik NOC, total jumlah kelas, atribut, operasi, asosiasi, dan generalisasi.

Actual Result	SUMIT menampilkan hasil ikhtisar berupa hasil pengukuran *understandability*, porsi *coupling*, porsi *cohesion*, porsi *inheritance*, nilai metrik NAssoc, metrik NA, metrik NOC, total jumlah kelas, atribut, operasi, asosiasi, dan generalisasi.
Status	Valid

6.4.8. Pengujian Validasi Banding *Design*

Berikut ini merupakan pengujuan validasi untuk kebutuhan Banding *Design*. Pada Tabel 6.32 merupakan hasil pengujian validasi *main flow* Banding *Design*, pada Tabel 6.33 merupakan hasil pengujian validasi *alternative flow* 1 Banding *Design*, pada Tabel 6.34 merupakan hasil pengujian validasi *alternative flow* 2 Banding *Design*, pada Tabel 6.35 merupakan hasil pengujian validasi *alternative flow* 3 Banding *Design*, dan pada pada Tabel 6.36 merupakan hasil pengujian validasi *alternative flow* 4 Banding *Design*.

Tabel 6.32 Pengujian Validasi *Main Flow* Banding *Design*

Kode SRS	SRS_F_SUMIT_008
Use case	Banding *Design*
Tujuan Pengujian	Memastikan SUMIT dapat memenuhi jalur utama kebutuhan fungsional Banding *Design*
Data Input	Berkas XML dengan struktur *simple*
Prosedur Pengujian	1. Pengguna membuka halaman membandingkan rancangan 2. Pengguna menekan tombol *browse* 3. Pengguna memilih sebuah berkas XML rancangan perangkat lunak berstruktur *simple* untuk dibandingkan 4. Pengguna menekan tombol *open* 5. Pengguna menekan tombol *compare*
Expected Result	SUMIT menampilkan perbandingan tingkat *understandability* antara dua buah rancangan perangkat lunak.
Actual Result	SUMIT menampilkan perbandingan tingkat *understandability* antara dua buah ancangan perangkat lunak.
Status	Valid

Tabel 6.33 Pengujia Validasi *Alternative Flow* 1 Banding *Design*

Kode SRS	SRS_F_SUMIT_008
Use case	Banding *Design*
Tujuan Pengujian	Memastikan SUMIT dapat memenuhi jalur alternatif pertama kebutuhan fungsional Banding *Design*
Data Input	-
Prosedur Pengujian	1. Pengguna membuka halaman membandinkan rancangan 2. Pengguna menekan tombol *compare*
Expected Result	SUMIT menampilkan pesan pastikan nilai pembanding ada
Actual Result	SUMIT menampilkan pesan pastikan nilai pembanding ada
Status	Valid

Tabel 6.34 Pengujian Validasi *Alternative Flow* 2 Banding *Design*

Kode SRS	SRS_F_SUMIT_008
Use case	Banding *Design*
Tujuan Pengujian	Memastikan SUMIT dapat memenuhi jalur alternatif kedua kebutuhan fungsional Banding *Design*
Data Input	Berkas non-XML
Prosedur Pengujian	1. Pengguna membuka halaman membandingkan rancangan 2. Pengguna menekan tombol *browse* 3. Pengguna memilih sebuah berkas non-XML untuk dibandingkan 4. Pengguna menekan tombol *open* 5. Pengguna menekan tombol *compare*
Expected Result	SUMIT menampilkan pesan berkas yang dipilih tidak menggunakan *format* XML
Actual Result	SUMIT menampilkan pesan berkas yang dipilih tidak menggunakan *format* XML
Status	Valid

Tabel 6.35 Pengujian Validasi *Alternative Flow* 3 Banding *Design*

Kode SRS	SRS_F_SUMIT_008
Use case	Banding *Design*
Tujuan Pengujian	Memastikan SUMIT dapat memenuhi jalur alternatiff ketiga kebutuhan fungsional Banding *Design*
Data Input	Berkas XML tidak berstruktur *simple*
Prosedur Pengujian	1. Pengguna membuka halaman membandingkan ancangan 2. Pengguna menekan tombol *browse* 3. Pengguna memilih sebuah berkas XML rancangan perangkat lunak tidak berstruktur *simple* 4. Pengguna menekan tombol *open* 5. Pengguna menekan tombol *compare*
Expected Result	SUMIT menampilkan pesan berkas XML yang dipilih tidak menggunakan struktur simple
Actual Result	SUMIT menampilkan pesan berkas XML yang dipilih tidak menggunakan struktur simple
Status	Valid

Tabel 6.36 Pengujian Validasi *Alternative Flow* 4 Banding *Design*

Kode SRS	SRS_F_SUMIT_008
Use case	Banding *Design*
Tujuan Pengujian	Memastikan SUMIT dapat memenuhi jalur alternatif keempat kebutuhan fungsional Banding *Design*
Data Input	-
Prosedur Pengujian	1. Pengguna membuka halaman membandinkan rancangan 2. Pengguna menekan tombol *compare*
Expected Result	SUMIT menampilkan pesan tidak ada berkas yang dipilih
Actual Result	SUMIT menampilkan pesan tidak ada berkas yang dipilih
Status	Valid

6.4.9. Pengujian Perhitungan Efisiensi

Pengujian perhitungan efisiensi dilakukan untuk mengetahui perbandingan efisiensi di antara perhitungan manual dan penggunaan SUMIT dalam mengukur tingkat *undestandability* rancangan perangkat lunak. Pengujian menggunakan 4 buah *task* dengan uji coba 2 berkas rancangan (sederhana dan kompleks) yang

dilakukan oleh seorang pengguna. Kelima *task* yang dibandingkan diantaranya, mengukur *coupling* dan porsinya, mengukur *cohesion* dan porsinya, mengukur inheritance dan porsinya, dan mengukur *understandability*. Berkas yang digunakan dalam pengujian ini merupakan rancangan perangkat lunak *open source* yang didapatkan dari hasil *reverse engineering* menggunakan *NetBeans UML Diagram Project*. Tingkat efisiensi diukur sesuai persamaan (2) dan persamaan (3). Berikut pada Tabel 6.37 dan Tabel 6.38 menunjukan hasil pengujian efisiensi:

Tabel 6.37 Hasil Pengujian Efisiensi Berkas 1 (Sederhana)

Task	Manual (dalam detik)	*Completion status*	SUMIT (dalam detik)	*Completion status*
Mengukur *coupling* dan porsinya berkas I	11	1	11	1
Mengukur *cohesion* dan porsinya berkas I	8	1	11	1
Mengukur *inheritance* dan porsinya berkas I	8	1	11	1
Mengukur *understandability* berkas I	21	1	11	1
Time based efficiency	0,139 *task*/detik		0,091 *task*/detik	
Overall relative efficiency	100%		100%	

Tabel 6.38 Hasil Pengujian Efisiensi Berkas 2 (Kompleks)

Task	Manual (dalam detik)	*Completion status*	SUMIT (dalam detik)	*Completion status*
Mengukur *coupling* dan porsinya berkas II	17	1	10	1
Mengukur *cohesion* dan porsinya berkas II	112	1	10	1
Mengukur *inheritance* dan porsinya berkas II	9	1	10	1
Mengukur *understandability* berkas II	25	1	10	1
Time based efficiency	0,052 *task*/detik		0,1 *task*/detik	
Overall relative efficiency	100%		100%	

Berdasarkan hasil yang ditunjukan pada tabel 6.37 untuk berkas I dan Tabel 6.38 untuk berkas II pengguna berhasil menyelesaikan keseluruhan *task* dengan benar. Dari hasil tersebut ditemukan hasil efisiensi berdasarkan *time based efficiency* untuk berkas I (rancangan sederhana) sebesar 0,139 *task*/detik ketika menggunakan cara manual dan 0,091 *task*/detik ketika menggunakan SUMIT. Nilai tersebut berarti efisiensi dengan cara manual lebih baik dari pada menggunakan SUMIT. Sedangkan untuk *overall relative efficiency* cara manual dan menggunakan SUMIT sama-sama mengahasilkan nilai 100%. Dan untuk berkas II (rancangan kompleks) diperoleh hasil efisiensi berdasarkan *time based efficiency* sebesar 0,052 *task*/detik ketika menggunakan cara manual dan 0,1 *task*/detik ketika menggunakan SUMIT. Nilai tersebut berarti efisiensi dengan menggunakan SUMIT lebih baik dari pada menggunakan cara manual. Sedangkan untuk *overall relative efficiency* cara manual dan menggunakan SUMIT sama-sama mengahasilkan nilai 100%. Dari hasil yang peroleh dapat disimpulkan bahwa efisiensi otomatisasi perhitungan understandability menggunakan SUMIT telah memenuhi kebutuhan non-fungsional sistem yang telah diuraikan dimana efisiensi pekerjaan dengan menggunakan SUMIT dapat dilakukan kurang dari 20 detik/pekerjaan atau tepatnya SUMIT mampu menyelesaikan 1 pekerjaan perhitungandalam waktu 10 detik untuk berkas peracangan yang sederhana maupun yang kompleks. Dan dari hasil pada Tabel 6.37 dan Tabel 6.38 menunjukan pengguna 100% dapat menyelesaikan pekerjaanya ketika menggunakan SUMIT.

6.5. Analisis Hasil

Berikut merupakan analisis hasil korelasi nilai *understandability* yang diperoleh ketika menggunakan SUMIT terhadap nilai *maintainability* yang telah diketahui dari sebuah perangkat lunak. Analisis menggunakan 10 data proyek perangkat lunak yang telah diketahui nilai *maintainability*-nya (Genero & Mario, 2001). Pengukuran korelasi menggunakan *spearman's rank correlation* (r_s). Maka diperoleh hasil koefisien korelasi sebesar 0,987 yang berarti keduanya memiliki korelasi yang sangat kuat. Hasil lebih rinci dapat dilihat pada Tabel 6.39:

Tabel 6.39 Korelasi Understandability SUMIT dengan Maintainability

Proyek	NA	NAssoc	NOC	*Understandability* SUMIT	*Maintainability*	$\sum d^2$
N1	4	1	0	1,65	1	0,38
N2	7	3	0	2,05	2	0,003
N3	14	1	0	2,11	2	0,012
N4	6	2	0	1,87	2	0,017
N5	8	3	0	2,09	3	0,83
N6	10	2	2	2,74	3	0,07
N7	9	1	0	1,88	2	0,14
N8	14	2	2	2,92	3	0,006

N9	18	2	4	3,79	3	0,62
N10	22	3	2	3,42	3	0,18
r_s						0,987

Berdasarkan Tabel 6.39 selisi ($\sum d^2$) antara *understandability* yang berhasil diukur oleh SUMIT dibandingkan dengan nilai maintainability yang telah diketahui tidak memiliki perbedaan yang signifikan. Hal tersebut mengakibatkan koefisien korelasi dari *spearman's rank* menunjukan hasil keterkaitan yang sangat kuat yaitu 0,987. Dari hasil tersebut menandakan bahwa meningkatnya nilai *understandability* turut mempengaruhi meningkatnya nilai *maintainability*. Hasil ini menegaskan agar dalam pengembangan perangkat lunak kita harus lebih menginvestasikan waktu kita pada fase perancangan demi menciptakan rancangan perangkat lunak dengan understandability yang baik agar memudahkan maintenance nantinya. Selain itu hasil ini juga memberikan kesimpulan bahwa nilai hasil perhitungan SUMIT dapat dijadikan salah satu acuan untuk memperkirakan usaha melakukan *maintenance* pada tahap perancangan perangkat lunak.

BAB 7 PENUTUP

7.1 Kesimpulan

Dari hasil penelitian pengembangan aplikasi perhitungan nilai *understandability* berdasarkan rancangan perangkat lunak diperoleh kesimpulan sebagai berikut:

1. Pengembangan aplikasi perangkat lunak SUMIT dikembangkan sesuai dengan metode yang diterapkan pada *waterfall model*. Model ini sesuai karena dalam penelitian ini seluruh kebutuhan telah dipahami dengan baik dan tidak terjadi perubahan yang radikal selama pengembangan sistem. Dalam penelitian ini dari tahap rekayasa kebutuhan dilakukan mulai dari proses pengkajian kepustakaan dan kemudian dilakukan elisitasi dengan cara pembuatan persona yang menghasilkan 8 buah kebutuhan fungsional dan 1 buah kebutuhan non-fungsional.
2. Pada tahap perancangan dapat menghasilkan perancangan arsitektur, perancangan algoritme dan antarmuka pengguna. Dari perancangan arsitektur mengasilkan 9 *sequence diagram* dan *class diagram* yang memuat 5 kelas boundry, 1 buah *interface*, 1 buah *abstract class*, 1 kelas *entity*, dan 5 buah kelas *controller* serta 3 buah kelas yang merepresentasikan penggunaan API/*Library* Swing. Sedangkan pada tahap implementasi berhasil menerapkan rancangan yang dibuat pada tahap perancangan diantaranya arsitektur, algoritme, dan antarmuka pengguna. Dalam implementasi SUMIT menggunakan API seperti Swing berhasil menerapkan antarmuka sistem dengan AWT sebagai *event-handler,* dan penerapan JAXP dinilai berhasil sebagai XML *file processor* untuk melakukan *parsing* dokumen XML dalam rangka pengukuran *understandability* rancangan perangkat lunak.
3. Pada tahap pengujian dilakukan kedalam tiga tahapan yaitu pengujian unit, pengujian integrasi, dan pengujian validasi. Pengujian unit dengan menggunakan teknik *basis path* berhasil menguji semua jalur uji, sedangkan pengujian integrasi dengan menggunakan *top-down testing* memberi hasil bahwa semua modul telah terintegrasi sesuai fungsinya, dan pengujian validasi dilakukan dengan teknik *equivalence partitioning* berhasil memvalidasi seluruh kebutuhan baik fungsional yang berjumlah 8 buah.
4. Dari pengujian efisiensi kinerja SUMIT menyatakan bahwa SUMIT mampu menyelesaikan 1 pekerjaan perhitungan dalam kurun waktu 10 detik dan pengguna 100% dapat menyelesaikan seluruh pekerjaanya ketika menggunakan SUMIT. Hasil ini menunjukkan bahwa kebutuhan non-fungsional berhasil dipenuhi oleh sistem.
5. Hasil perhitungan *understandability* yang didipatkan ketika menggunakan SUMIT memiliki koefisien korelasi sebesar 0,987 terhadap *maintainability* yang berarti sangat kuat damana menegaskan bahwa dalam pengembangan perangkat lunak disarankan untuk menginvestasikan lebih pada fase

perancangan agar menghasilkan rancangan yang mudah untuk di*maintain* serta menandakan hasil dari SUMIT dapat dijadikan salah satu acuan untuk mengetahui *understandability* dalam memperkirakan usaha untuk melakukan *maintenance* yang dapat dilakukan sedini mungkin.

7.2 Saran

Saran untuk penulisan pengembangan aplikasi perhitungan nilai *understandability* berdasarkan rancangan perangkat lunak adalah sebagai berikut:

1. Metrik-metrik yang dilibatkan dalam perhitungan persamaan *multivariate undestandability metric* masih menggunakan metrik versi Chidamber dan Kemerer serta Lorenz dan Kidd yang masih bisa ditingkatkan menggunakan metrik-metrik yang lebih mutakhir.
2. Sistem masih tidak fleksibel dalam melakukan *parsing* berkas xml berisi *class diagram*. Berkas yang mampu di*parsing* hanya yang menggunakan bentuk standar dari Visual Paradigm sehingga masih bisa dikembangkan agar dapat *compatible* dengan standar xml yang lain atau mengikuti standar xml yang lebih umum.
3. Selain *understandability* SUMIT dapat dilengkapi lagi untuk parameter lainnya yang mampu memperkirakan tingkat *maintainability* pada saat fase perancangan perangkat lunak.

DAFTAR PUSTAKA

Alhumaidan, F. (2012). A Critical Analysis and Treatment of Important UML Diagrams Enhancing Modeling Power. *Intelligent Information Management*, 231-237.

Ashdown, L., Greenberg, J., & Melnick, J. (2014). *Oracle XML Developer's Kit Programmer's Guide 11.1 Release.* Oracle.

Bennett, S., McRoob, S., & Farmer, R. (2011). *Object-Oriented Systems Analysis and Design Using UML4th Edition.* New York: McGraw-Hill.

Chidamber, S., & Kemerer, C. (1994). A Metric Suite for Object Oriented Design. *IEEE Transaction on Software Engineering*, 467-493.

Cole, B., dkk. (2002). *Java Swing, 2nd Edition.* Sebastopol: O'Reilly.

Genero, M., & Mario, P. (2001). A Controlled Experiment for Corraborating The Usefulness of Class Diagram Metrics. *International Journal of Multimedia and Ubiquitous Engineering*, 369-376.

Hamdy, K., Elsoud, M., & El-Halawany, A. (2011). UML-Web Engineering Framework for Modeling Web Application. *Journal of Software Engineering*, 49-63.

Isaias, P., & Issa, T. (2015). *High Level Models and Methodologies for Information Systems.* New York: Springer.

Izadkhah, H., & Hooshyar, M. (2017). Class Cohesion Metric for Software Engineering: A Critical Review. *Computer Science Journal of Moldova*, 788-804.

Kumar, D. S., & Prasad, R. (2015). New Metrics for System Understandability of Inheritance Hierarchies. *International Journal of Research Studies in Computer Science and Engineering*, 59-62.

Kurniawan, T. A. (2018). Pemodelan Use Case (UML): Evaluasi Terhadap Beberapa Kesalahan Dalam Praktik. *Jurnal Teknologi Informasi dan Ilmu Komputer (JTIIK)*, 77-86.

Nazir, M., Khan, R. A., & Mustafa, K. (2010). A Metric Based Model for Understandability Quantification. *Journal of Computing*, 90-94.

Pressman, R. S. (2010). *Software Engineering: A Practitioner's Approach 7th Edition.* New York: McGraw-Hill.

Rajnish, K. (2014). Class Complexity Metric to Predict Understandability. *International Journal of Information Engineering and Electronic Business*, 69-76.

Royce, W. (2011). Managing the Development of Large Software Systems: Concepts and Techniques. Dalam I. Sommerville, *Software Engineering 9th Edition* (hal. 30). Boston: Addison-Wesley.

Seergev, A. (2010). *Efficiency.* Diambil kembali dari User Interfaces Design and UX/Usability Evaluation: designer.net/usability/efficiency.htm

Singh, N., & Tripathi, A. (2015). A Design Phase Understandability Metric Based on Coupling and Cohesion for Object-Oriented Systems. *Advances in Intelligent Systems and Computing*, 321-325.

Sommerville, I. (2011). *Software Engineering 9th Edition.* Boston: Addison-Wesley.

Sullivan, G. (2010). *Operation & Maintenace Best Practice 3.0.* Washington, DC: FEMP.

Uchida, S., & Shima, K. (2004). An Experiment of Evaluating Software Understandability. *Journal of Systemics, Cybernetics and Informatics*, 7-11.

Yadav, A., & Khan, R. (2011). Class Cohesion Complexity Metric (C3M). *International Conference on Computer & Communication Technology (ICCCT)*, 363-366.

Zar, J. H. (2005). Spearman Rank Correlation. Dalam P. Armitage, & T. Colton, *Encyclopedia of Biostatistics 2nd Edition* (hal. 5095-5101). Hoboken: John Wiley & Sons.